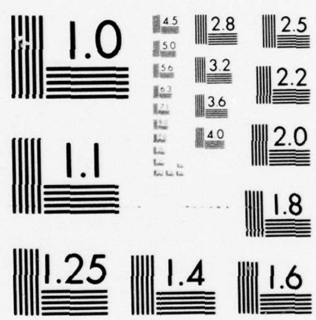
AD-A040 103 GENERAL RESEARCH CORP SANTA BARBARA CALIF F/G 9/2
JAVS TECHNICAL REPORT. USER'S GUIDE.(U)
APR 77 C GANNON, N B BROOKS, R J URBAN F30602-76-C-0233
UNCLASSIFIED RADC-TR-77-126-VOL-1 NL

1 OF 2
AD
A040103


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

RADC-TR-77-126, Volume I (of three)
Final Technical Report
April 1977

JAVS TECHNICAL REPORT
User's Guide

General Research Corporation

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441

Some of the pages of this report are not of the highest printing quality but because of economical consideration, it was determined in the best interest of the government that they be used in this publication.

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

This report has been reviewed and approved for publication.

APPROVED: Frank S La Monica

FRANK S. LA MONICA
Project Engineer

APPROVED: Robert D. Krutz

ROBERT D. KRUTZ, Col, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

Do not return this copy. Retain or destroy.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

AMERICAN REVOLUTION BICENTENNIAL 1776-1976

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-77-126, Volume I (of three)	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) JAVS TECHNICAL REPORT User's Guide		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report May 76 - Nov 76
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) C. Gannon N. B. Brooks R. J. Urban		8. CONTRACT OR GRANT NUMBER(s) F30602-76-C-0233
9. PERFORMING ORGANIZATION NAME AND ADDRESS General Research Corporation P. O. Box 3587 Santa Barbara CA 93105		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 63728F 55500838
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIM) Griffiss AFB NY 13441		12. REPORT DATE April 1977
		13. NUMBER OF PAGES 99
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same		
18. SUPPLEMENTARY NOTES RADC Project Engineer: Frank La Monica (ISIM)		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Computer Software, Software Testing, Software Verification, JAVS, Automated Verification System.		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The JOVIAL Automated Verification System (JAVS) is a tool for analyzing source programs written in the J3 dialect of the JOVIAL language. From the user's viewpoint, JAVS consists of a sequence of processing steps which (1) analyze his JOVIAL source text, (2) guide him in preparing test cases for his code, (3) analyze the results of tests executed by his code, and (4) automatically document his code. The purpose of this document is to introduce the tester to JAVS and to the		

402 754

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

over

process of software testing supported by JAVS. The information provided in this guide on JAVS usage is intentionally limited to the beginning user. The appendices provide the information necessary for operating JAVS at RADC and can be referenced by the sophisticated as well as the beginning user. The information presented on the testing methodology which JAVS supports is applicable to both the beginning and sophisticated user of JAVS.

PREFACE

The purpose of this guide is to introduce the user into the realm of automated testing. Software testing supported by an automated verification system requires knowledge of two inseparable factors: the verification tool and the testing methodology which the tool supports. The information concerning the usage of JAVS is intentionally limited to the beginning user. The only prerequisite information is a knowledge of the JOVIAL language. The casual user should have good success in analyzing the behavior of his programs using the description of JAVS capabilities and a few commands set forth in this guide. All job control and file information is presented in appendixes, along with estimates of processing time and core requirements.

The testing methodology which JAVS supports is described in this guide because of its importance to JAVS users at all levels of expertise. Although there is no single general methodology which applies to all testing situations, there are a number of important issues that even the beginning verification tool user should recognize in order to make the testing experience successful. Section 10 of this guide focuses on software preparation for JAVS-supported testing, testing goals, resources required, and testing strategy.

In the series of JAVS reports, this guide should be read first. The information presented should enable the tester to become a new user of JAVS at RADC. Once the user has experienced some of the capabilities that JAVS offers, the <u>JAVS Reference Manual</u>[1] should be used to supply the complete details of JAVS features and command language.

For more comprehensive treatment of software testing methodology not restricted solely to JAVS-supported testing, the reader is referred to the <u>Methodology Report</u>.[2] This report describes experiences with using current Automated Verification Systems, approaches to software quality, and advanced AVS capabilities.

LIST OF JAVS REPORTS

- JAVS Technical Report: Vol. 1, User's Guide. This report is an introduction to using JAVS in the testing process. Its primary purpose is to acquaint the user with the innate potential of JAVS to aid in the program testing process so that an efficient approach to program verification can be undertaken. Only the basic principles by which JAVS provides this assistance are discussed. These give the user a level of understanding necessary to see the utility of the system. The material on JAVS processing in the report is presented in the order normally followed by the beginning JAVS user. Adequate testing can be achieved using JAVS macro commands and the job streams presented in this guide. The Appendices include a summary of all JAVS commands and a description of JAVS operation at RADC with both sample command sets and sample job control statements.

- JAVS Technical Report: Vol. 2, Reference Manual. This report describes in detail JAVS processing and each of the JAVS commands. The Reference Manual is intended to be used along with the User's Guide which contains the machine-dependent information such as job control cards and file allocation. Throughout the Reference Manual, modules from a sample JOVIAL program are used in the examples. Each JAVS command is explained in detail, and a sample of each report produced by JAVS is included with the appropriate command. The report is organized into two major parts: one describing the JAVS system and the other containing the description of each JAVS command in alphabetical order. The Appendices include a complete listing of all error messages directly produced by JAVS processing.

- JAVS Technical Report: Vol. 3, Methodology Report. This report describes the methodology which underlies and is supported by JAVS. The methodology is tailored to be largely independent of implementation and language. The discussion in the text is intended to be intuitive and demonstrative. Some of the methodology is based upon the experience of using JAVS to test a large information management system. A long-term growth path for automated verification systems that supports the methodology is described.

- JAVS Computer Program Documentation: Vol. 1, System Design and Implementation. This report contains a description of JAVS software design, the organization and contents of the JAVS data base, and a description of the software for each JAVS component: its function, each of the modules in the component, and the global data structures used by the component. The report is intended primarily as an informal reference for use in JAVS software maintenance as a companion to the Software Analysis reports described below. Included in the appendices are the templates for probe code inserted by instrumentation processing for both structural and directive instrumentation and an alphabetical list of all modules in the system (including system routines) with the formal parameters and data type of each parameter.

- JAVS Computer Program Documentation: Vol. 2, Software Analysis. This volume is a collection of computer output produced by JAVS standard processing steps. The source for each component of the JAVS software has been analyzed to produce enhanced source listings of JAVS with indentation and control structure identification, inter-module dependence, all module invocations with formal and actual parameters, module control structure, a cross reference of symbol usage, tree report for each leading module, and report showing size of each component. It is intended to be used with the System Design and Implementation Manual for JAVS software maintenance. The Software Analysis reports, on file at RADC, are an excellent example of the use of JAVS for computer software documentation.

- JAVS Preprocessor for JOVIAL. This report, prepared for GRC by its subcontractor, System Development Corporation (SDC), describes the software for the JAVS-2 component: its origin as the GEN1 part of the SAM-D ED Compiler, the modifications made in GEN1 to adapt the code for JAVS-2, the JAVS-2 code modules, and the data structures. It contains excerpts of other SDC reports on the SAM-D ED JOVIAL Compiler System. The report reflects the status of the software for JAVS-2 as delivered by SDC to GRC in September 1974. The description of JAVS-2 software contained in the System Design and Integration report reflects the status of JAVS-2 as delivered to RADC by GRC in September 1975 and thereby supercedes the SDC report.

- JAVS Final Report. The final report for the project describes the implementation and application of a methodology for systematically and comprehensively testing computing software. The methodology utilizes the structure of the software undergoing test as the basis for anlaysis by an automated verification system (AVS). The report also evaluates JAVS as a tool for software development and testing.

CONTENTS

CONTENTS (CONT.)

ILLUSTRATIONS

TABLES

EVALUATION

The purpose of this effort, identified in TPO V/4.2, was to enhance the JOVIAL Automated Verification System (JAVS) and to implement a systematic software testing program using the JAVS to assist in the testing process. Developed to aid in the testing and verification of JOVIAL J3 programs, it provides the ability to increase the practical reliability of software by increasing the achieved level of testing.

As a result of this effort, the JAVS was successfully enhanced and tuned for operational use. Included in its excellent supporting documentation is a refined methodology for testing large, complex software systems.

Frank S. La Monica

FRANK S. LA MONICA
Project Engineer

1. INTRODUCTION

JAVS was developed as a tool to aid JOVIAL software developers and testers determine the extent to which their programs have been tested and to assist in deriving additional test cases to verify the software. Up to now, testing has been without an orderly approach and without accurate means to determine exactly what portions of code have been exercised. JAVS provides a testing approach[2] and an automated tool for measuring the effectiveness of test data in terms of program structure.

1.1 USER'S GUIDE ORGANIZATION

Section 1 of this guide introduces JAVS in the validation of JOVIAL software. The overview description includes JAVS capabilities and limitations in order to provide an assessment of what tasks are automated and what tasks the software tester must undertake using JAVS reports as a guide. Section 1 also contains a description of JAVS's organization in terms of the tasks it performs and the accomplishment of the tasks through user commands.

Section 2 describes the utilization of JAVS: what preliminary steps need to be taken and how to use the command language. Sections 3 through 8 describe more specifically the processing steps taken in a typical validation effort. Section 9 summarizes the commands, and Section 10 describes a testing strategy. At this point the reader can become a JAVS user and consider a test plan for his particular software. The appendixes will be needed to operate JAVS at RADC. Appendix A contains a summary of all JAVS commands; Appendix B contains the expansion of the macro commands; Appendix C describes the files used by JAVS command sets; Appendix E contains estimations of processing time and file size requirements, Appendix F contains the JAVS installation requirements for RADC.

1.2 JAVS CAPABILITIES

Before proceeding, it should be made clear what JAVS can and cannot do. As a testing tool, JAVS provides trace and coverage reports showing program behavior during a test. Tracing can be performed, at user option, to show module invocations and returns or to show which outcome was taken for each conditional operation in the program. In addition, the user can trace "important" events, such as overlay link loading, by invoking one of the JAVS data collection routines. Test performance coverage reports showing statements and/or decision outways can be obtained on a per-module, per-test-case, and per-test-run basis. These reports allow the user to focus on untested modules, program paths, and statements.

If the testing target is determined to be a set of modules which received little or no coverage during the test execution, JAVS reports can be obtained to list all invocations (and the statement numbers of the calls) to the modules and to show the modules' interactions with the rest of the system in terms of calling trees and interaction matrices. If the testing target is a segment of code within a module, the user can request a JAVS report showing the statements that lead up to the target. Armed with this "reaching set" report, the user can spot key variables whose values affect the flow through the program paths and locate all instances of the variables in the system-wide cross reference.

Retesting may necessitate code changes in some of the modules in the system to remove dead code or coding errors found during the test analysis. To facilitate determining all modules in the system which could be affected by the code changes, a JAVS report will show the interaction between the selected set of modules and the rest of the system.

JAVS uses a data base to store information about the test program. The availability and management of this information form the basis for a variety of services in addition to the primary task of testing assistance. Computer program documentation, debugging through JAVS computation directives, and reports useful for code optimization are the major side benefits of JAVS.

Computer documentation requirements for the Air Force typically specify flow charts and lists of program variables and constants. In the JAVS development and implementation contracts these requirements were replaced by specifying certain JAVS reports, i.e. self-documentation. It was found that the module listings (enhanced by indentation and identification of decision points), module control flow pictures, module invocation reports (showing formal and actual parameter lists), module interdependence reports, and a cross-reference report for each JAVS component are more meaningful documentation and are generated automatically by JAVS.

Software development can be assisted by using JAVS to document and test the system as it is built. To aid in data flow analysis and checking of array sizes and variable execution values, JAVS offers computation directives. The directives are a special form of JOVIAL comment, recognized by JAVS and expanded into executable code (using the JOVIAL monitor statement) during the instrumentation phase. The user can check logic expressions with an ASSERT directive, check boundaries of selected variables with an EXPECT directive, and turn on and off the standard monitor tracing with TRACE and OFFTRACE directives. The computation directives are described in the Reference Manual.

Code optimization is aided by the post-test reports, which show the number of times each statement is executed and the execution time (in C.P. milliseconds) spent in the modules. Modules which are never called and should be removed are listed in another JAVS report.

1.3 JAVS LIMITATIONS

Testing coverage results indicate what parts of the program were executed. It is up to the user to determine if the program's output is reasonable. One of JAVS post-test analysis reports lists the execution coverage during the test run in terms of the percentage of decision outways taken. A decision outway (decision-to-decision path, or DD-path) is the set of statements executed as the result of the evaluation of a predicate (conditional operation). A good standard for the testedness of a program is to exercise every decision outway at least once. This level of testing is more rigorous than testing every program statement at least once. However, it should be emphasized that certain combinations of DD-paths may contain errors which are not detected in merely executing each outway one time.

1.4 JAVS ORGANIZATION

JAVS reads the user's JOVIAL program as data and performs syntax, structural, and instrumentation analyses on the source code. JAVS communicates with the user through a command language and utilizes a data base to store the information about the program. The user is provided with an instrumented file of the selected program modules with which the user supplies test data for execution. The execution results are written to a file from which JAVS's post-test analyzer issues execution tracing and coverage reports.

Six functional processes, in addition to execution with test data, make up the substance of software validation provided by JAVS. The organization of JAVS is defined by these six tasks. To reduce the burden of the user, JAVS exists as an overlay program at RADC with a macro command language supplementing a large, versatile standard command language. The processing steps and their basic functions are listed below:

<u>BASIC, Source Text Analysis</u>: Source text input, lexical analysis, and initial source library creation

<u>STRUCTURAL, Structural Analysis</u>: Structural analysis and execution path identification; library update with structure and path information

<u>INSTRUMENT, Module Instrumentation</u>: Program instrumentation for path coverage analysis and program performance directed by user; library update with probe test instrumentation

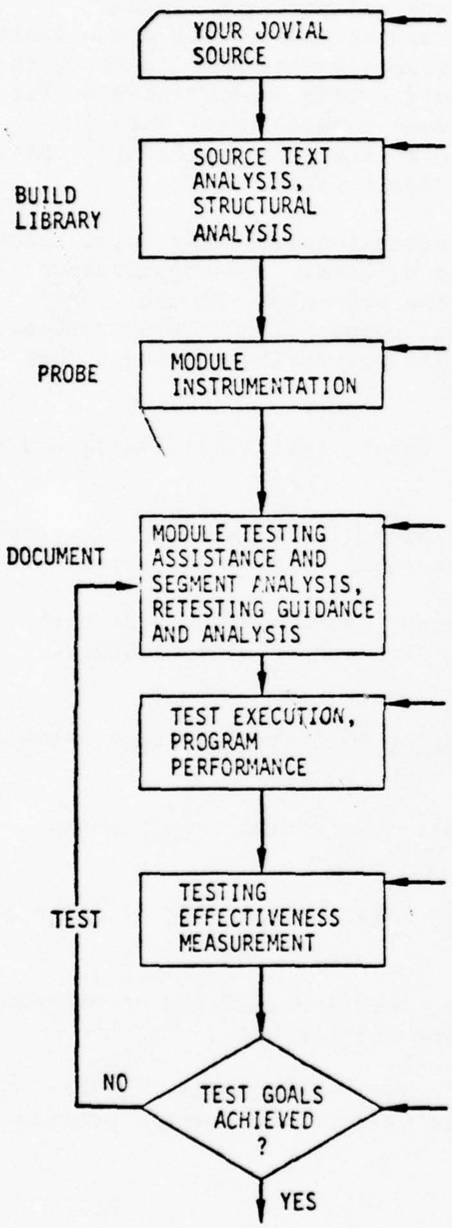
<u>ASSIST, Module Testing Assistance and Segment Analysis</u>: Testing assistance for improved program coverage

<u>DEPENDENCE, Retesting Guidance and Analysis</u>: Retesting requirements analysis for changed modules

<u>TEST EXECUTION</u>: Execution of instrumented code and analysis of directed program performance

<u>ANALYZER, Test Effectiveness Measurement</u>: Detailed analysis of program path coverage; execution traces and summary statistics

These steps need not be performed in the above order; other orders may be preferable at times. An overview of how JAVS is used in the testing process is shown in Fig. 1.1.



JOVIAL source code is input for processing and analysis. A special form of comment (optional) inserted by the user directs JAVS processing for program performance analysis.

AN-47751

JAVS analyzes the code and generates a directed graph of the control structure.
The possible flows through the program are determined. All pertinent data is stored in a data base for later use. Additional or changed source code causes an existing data base to be updated.

JAVS automatically inserts software probes into the source code to intercept and record program flow during execution. A second type of instrumentation is used to record statistics on program performance according to directives inserted by the user in the source.

JAVS provides a variety of services which assist in the establishing of a continuing testing strategy both at the detailed level within a module and at the module interdependence level. Test cases are constructed by the user. Reports can be used as program documentation.

Program execution provides normal computational results as well as outputs from the instrumentation. Structural flow output is recorded for later post-test analysis by JAVS. User-directed program performance output is interspersed with normal program output.

JAVS includes detailed post-test analysis facilities which provide measures of testing thoroughness, both individually and cumulatively for a set of test cases.

The results are examined by the user to determine if test goals have been met and testing is completed.

Figure 1.1. Overview of the JAVS in the Testing Process

2 USING JAVS

The process of program verification is best described by example. One purpose of this User's Guide is to present an overview of JAVS capabilities through example programs processed by the JAVS execution steps. It is important to note that while there are six processing steps a given validation effort may require use of only a few of these. The selection of appropriate processes is largely a user decision, based upon his requirement for the information that the various steps provide. As each step is described, through example, the user will gain insight into its utility for his particular needs. In order to develop a basic understanding of the processing sequences to be utilized in the examples, Fig. 2.1 illustrates the potential JAVS processing flows in terms of step interdependencies.

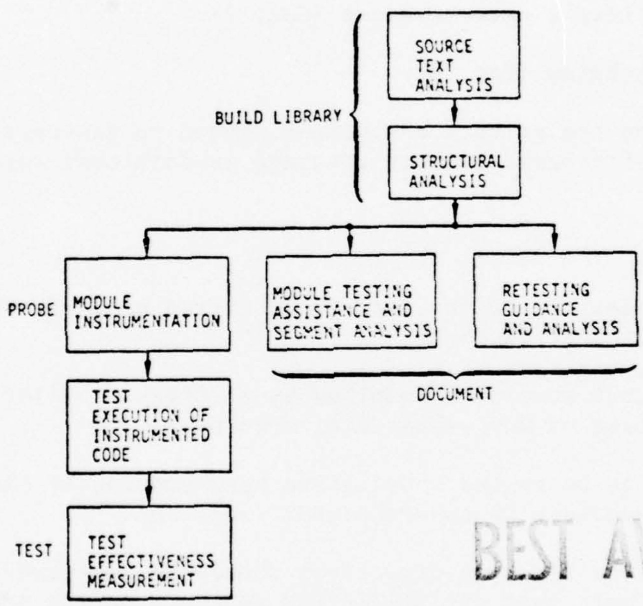


Figure 2.1. JAVS Processing Sequences

The user must provide three major types of input to JAVS: (1) the source code to be tested, (2) a set of commands to direct JAVS processing, and (3) test data for program execution. Section 2.2 describes the preparation of the source code for input to JAVS. Section 2.3 describes the rules for inputting commands.

2.1 TYPICAL PROCESSING SEQUENCE

This guide is organized to lead the user through the following sequence of steps:

1. Build a data base library containing source text and structural analyses (Sec. 3).
2. Document the source text (Sec. 4).
3. Instrument the modules (Sec. 5).
4. Execute the program (Sec. 6).
5. Measure the test's effectiveness (Sec. 7).
6. Retest the program (Sec. 8).

These steps provide the primary assistance needed to generate test cases and measure the extent of program testing coverage as each test case is input to the system.

2.2 PRELIMINARY STEPS

Before the source text to be verified is submitted to JAVS, the user should take certain preliminary steps:

1. The source text should be compiled by a JOVIAL compiler to confirm that it is free of any syntactical errors.
2. The program to be tested should have been previously executed with test data necessary to ensure proper execution.
3. JAVS text identification directives should be inserted in the source if there is more than one START-TERM sequence in the program.
4. JAVS computation directives should be inserted in the source if the performance testing capability is utilized.

Both types of JAVS directives (a special form of JOVIAL comment) are described in detail in the JAVS Reference Manual[1]. The JAVS text identification directive is used to assign a unique name to a JOVIAL START-TERM sequence (program, sub-program, or COMPOOL). If no text directive is assigned to a START-TERM sequence, the text name *NOJAVS* is assigned as a default. The computation directives are used to make assertions about the behavior of the program and to verify the value of specified variables without altering the logic of the program. These directives can make valuable contributions in debugging and boundary conditions testing. It is suggested that the user acquire some familiarity with JAVS before utilizing the computation directives. The Reference Manual (Sec. 1.5) describes their capability and utilization.

The following sections describe the recommended sequence of step executions to be utilized by the beginning user. Although JAVS is capable of processing very large JOVIAL programs, we recommend that the tester select a modest program (several hundred JOVIAL statements or less) to use in his first experience with JAVS processing.

2.3 COMMAND STRUCTURE

The user directs JAVS processing by a set of commands. There are four "macro" commands which can be used with the JAVS 2.0 overlay program, in addition to a variety of standard commands. Each macro command expands into a set of commonly used standard JAVS commands. While both types of commands can be used together, the user is advised to be aware of the expansion of the macros before combining commands. Table 2.1 shows the relationship between macro commands, standard commands, and the processing tasks. Sections 3-8 describe each task, as well as the appropriate commands to use, and the process of executing the test program is described in Sec. 6.

All commands are input one per card. Blanks are ignored, so the commands are free-form. The card scan ends with a period or with the end of a card. If a command requires more than one card, a comma must appear at the last non-blank character of each card preceeding the continuation card. Up to three continuation cards may be used. Each command consists of a sequence of terms separated by a comma or an equals sign.

TABLE 2.1

RELATIONSHIP BETWEEN COMMANDS AND TASKS

Macro Command Keyword	Standard Command Keyword	Task
BUILD LIBRARY	BASIC STRUCTURAL	Syntax analysis Structural analysis
PROBE	INSTRUMENT	Structural and computation instrumentation
DOCUMENT	ASSIST PRINT DEPENDENCE	Module and inter-module reports
TEST	ANALYZER	Post-test coverage and trace analysis

3 PRIMARY ANALYSIS

Prior to instrumentation, documentation, testing, or retesting, a set of primary analyses must be performed. Syntax analysis is performed on the JOVIAL source program, transforming it into a format appropriate for storage on a random-access data base (library file). Using the information on the data base, structural analysis is performed on the executable modules, updating the tables in the data base library. Structural analysis includes building a directed program graph which is the basis for instrumentation and testing analyses. The subject matter of this section, primary analysis, is shown in the context of the testing process in Fig. 3.1.

3.1 TASKS

Syntax analysis consists of breaking-down each START-TERM sequence of the JOVIAL source text into invokable modules. A data base library is created containing internal tables representative of program text, statement descriptions and symbol classification.

Structural analysis adds to the data base library a description of program structure in terms of decision-to-decision paths. These paths represent a unique and systematic ordering of all decision outways. Figures 3.2 and 3.3 illustrate the concept of DD-paths. A DD-path consists of all the executable statements from a conditional statement to the next conditional statement. Figure 3.2 shows the statement membership for each DD-path in module EXAMPL. This module contains 12 DD-paths. Below each DD-path number (listed across the page) is the order in which the statements are placed on each DD-path. For example, DD-path 2 consists of statements 15, 16, 29, 30, and 31 in that order.

3.2 PRIMARY ANALYSIS INPUT

JAVS requires two input files for syntax analysis: the JOVIAL source program in BCD mode on file READER (09) and the JAVS commands in BCD mode on file COMMAN (05). If the source program contains more than one START-TERM sequence, or if the source text is a COMPOOL or requires a COMPOOL to compile, the user must insert a JAVS text identification directive as the first statement. This statement is described in Sec. 1.4 of the Reference Manual and is shown in Figs. 3.2 and 3.4.

Input for structural analysis are the JAVS commands and the data base library created during the syntax analysis.

3.3 COMMANDS

Primary analysis can be accomplished by the single JAVS command:

```
BUILD LIBRARY [=<name>].
```

This command expands into the following set of standard commands:

```
CREATE LIBRARY = <name>.  (default name is TEST)
START.
```

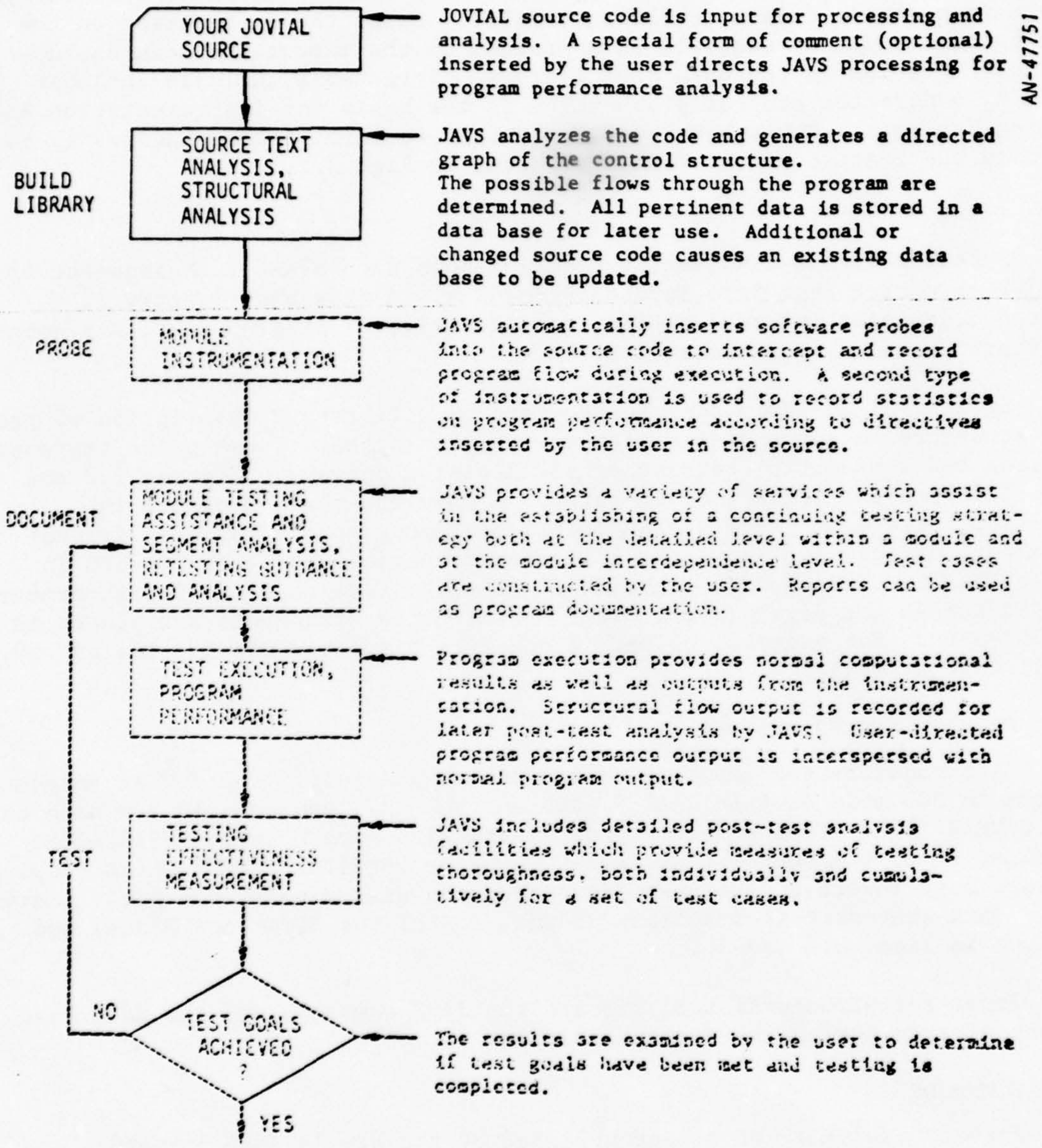
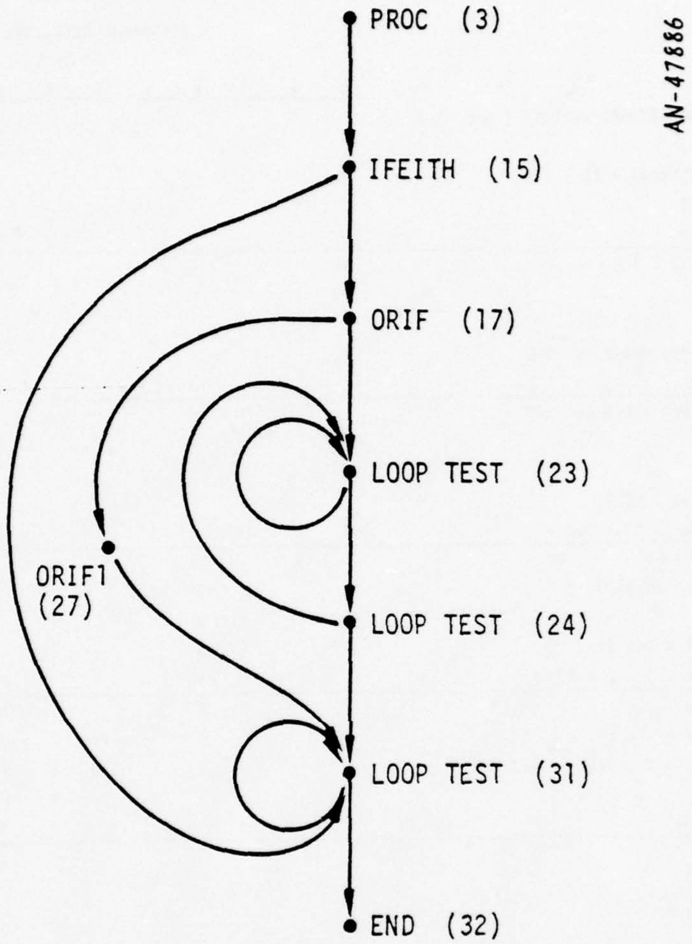


Figure 3.1. The Role of Primary Analysis in the Testing Process

		Statement Execution Order DD-Path											
		1	2	3	4	5	6	7	8	9	10	11	12
1	". JAVSTEXT EXAMPL COMPUTE (COMPOL)"												
2	START												
3	PROC EXAMPL (AA = BB) $	1											
4	ITEM AA F $												
5	ITEM BB F $												
6	ARRAY CC 2 2 F $												
7	BEGIN												
8	BEGIN												
9	+10.E-001 +10.E-001 END												
10	BEGIN												
11	+10.E-001 +10.E-001 END												
12	END												
13	BEGIN	2											
14	MONITOR BB , CC $												
15	IFEITH AA LS 00.E-001 $	3	1	1									
16	BB = - AA $		2										
17	ORIF AA EQ 00.E-001 $			2	1	1							
18	BEGIN				2								
19	BB = 00.E-001 $				3								
20	FOR I = 0 , 1 , 1 $				4				2				
21	BEGIN				5				3				
22	FOR J = 0 , 1 , 1 $				6		2		4				
23	CC ($ I , J $) = 00.E-001 $				7		1,3	1	5				
24	END							2	1	1			
25	RETURN $									2			
26	END												
27	ORIF 1 $					2					1		
28	BB = AA $										2		
29	END		3								3		
30	FOR K = 0 , 1 , 1 $		4								4	2	
31	CC ($ K , 0 $) = BB / 20.E-001 $		5							3	5	1,3	1
32	END												2
33	TERM $												

Figure 3.2. Example of Structural Analysis



$\xrightarrow{3}$	Numbers represent DD-paths
•	Dots represent decision points
IFEITH	Words represent decision statement types
(31)	Parenthetical numbers represent statement numbers

Figure 3.3. Pictorial Example of Structural Analysis

```
''.JAVSTEXT EXAMPL COMPUTE (COMPOL)'' *
  START
  PROC EXAMPL  (AA=BB)$
  ITEM AA F $
  ITEM BB F$
  ARRAY CC 2 2 F$
  BEGIN  BEGIN  1.0 1.0 END
         BEGIN  1.0 1.0 END  END
  BEGIN
  MONITOR BB, CC$
  IFEITH AA LS 0.0$
  BB = -AA$
  ORIF AA EQ 0.0$
  BEGIN
  BB = 0.0$
  FOR I=0,1,1$
  BEGIN
  FOR J=0,1,1$
  CC($I,J$) = 0.0$
  END
  RETURN$
  END
  ORIF 1$
  BB = AA$
  END
  FOR K=0,1,1$
  CC($K,0$) = BB/2.0$
  END
  TERM$

EXAMPL   (EXAMPL   ) COMPLETED
   ********    NO ERRORS WERE FOUND BY JAVS-2  ******
```

*This statement is necessary to inform JAVS that a COMPOOL is being used. See Sec. I.4 of the JAVS Reference Manual for description of the text identification statements.

Figure 3.4. BASIC Output

```
BASIC,COMMENTS = OFF.
BASIC.
FOR LIBRARY.
STRUCTURAL.
END FOR.
END.
```

The actions taken by the macro command (or equivalent set of standard commands) is to initialize the JAVS system with a library whose name is <name> (or TEST if none is specified), process syntax analysis (BASIC) removing JOVIAL comment statements, and perform structural analysis for all modules on the newly created library.

There are several BASIC processing options, all described in the Reference Manual. If the user wishes to exercise any of the options, to add the JOVIAL comments in the source text to the library, or to perform structural analysis on a subset of the modules, he cannot use the BUILD LIBRARY macro command; instead, the desired sequence of BASIC commands must be supplied. Section 5 of the Reference Manual contains sample command sets for each command description.

3.4 PRIMARY ANALYSIS OUTPUT

The main output is a data base library file containing the source text transformed into invokable modules and tables for other functional processing and reports. Printed output consists of the card image listing of the JOVIAL source code (this can be turned off with a BASIC option) along with JAVS error messages, if any, and a few descriptive lines for each module stating the number of DD-paths generated. If any syntax errors are printed adjacent to the offending source text line, they should be scrutinized. A complete list of JAVS errors is in Appendix B of the Reference Manual. Some errors will require source code changes before further processing, and some errors are syntactical warnings.

Figure 3.4 shows the syntax analysis output and Fig 3.5 shows structural analysis output for module EXAMPL.

```
JOVIAL AUTOMATED VERIFICATION SYSTEM   *** SECONDARY MODULE ANALYSIS ***

MODULE  EXAMPL  > OF JAVSTEXT <EXAMPL  >.
     MODULE DEPENDENCE TABLE CONSTRUCTED.
     STATEMENT DESCRIPTOR BLOCKS UPDATED.
     DD-PATH TABLE CONTAINS   12 ENTRIES.
```

Figure 3.5. STRUCTURAL Output

4 DOCUMENTATION

Automated documentation, showing inter- and intra-module relationships, is useful during the software development, testing, and maintenance stages. Figure 4.1 shows the JAVS documentation activity in the context of the testing process. JAVS provides a wide variety of reports at user request. Some of the reports pertain only to a selected module; others pertain to all modules on a library. Seven of the most commonly requested types of reports are generated by the macro command:

```
DOCUMENT.
```

This command expands into the following set of standard commands:

```
OLD LIBRARY = TEST.
START.
ASSIST,CROSSREF,LIBRARY.                    (Fig. 4.2)
DEPENDENCE,GROUP,LIBRARY.                   (Fig. 4.3)
DEPENDENCE,GROUP,AUXLIB.                    (Fig. 4.4)
DEPENDENCE,SUMMARY.                         (Fig. 4.5)
FOR LIBRARY.
PRINT,MODULE.                               (Fig. 4.6)
DEPENDENCE,BANDS=5.                         (Fig. 4.7)
DEPENDENCE,PRINT,INVOKES.                   (Fig. 4.8)
END FOR.
END.
```

This collection of reports provides a static analysis of the individual modules and of the interaction of the system of modules on the library. Section 4.2 describes alternate or additional reports, and Sec. 4.3 shows sample output of each report as well as a description of its utilization.

4.1 DOCUMENTATION INPUT

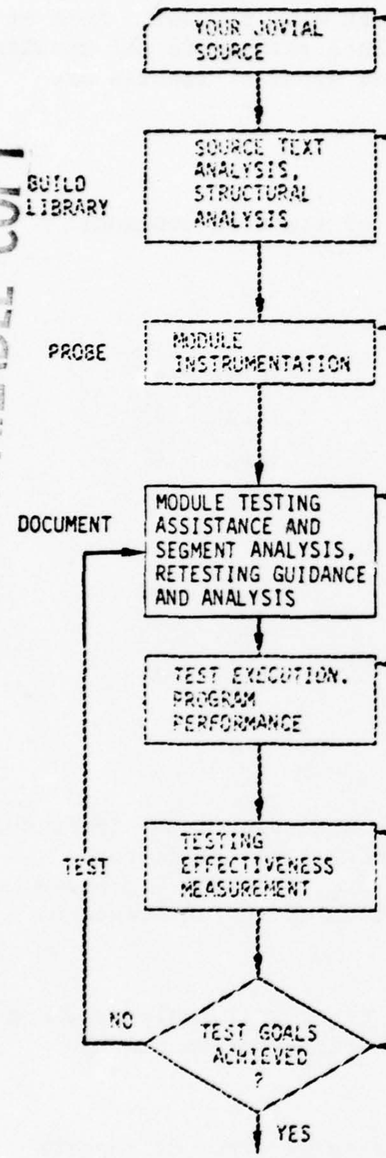
The data base library containing syntax and structural analyses along with JAVS commands are the input for the software documentation process.

4.2 COMMANDS

The single command DOCUMENT will provide the seven types of reports shown in Figs. 4.2-4.8. By mixing JAVS standard and macro commands, the user can specify any combination of reports. A selection of documentation command sets is:

- Obtain the three module documentation reports for a specified JAVSTEXT (named START-TERM sequence), in addition to the four library-wide reports.

```
DOCUMENT,JAVSTEXT = <textname>.
```



JOVIAL source code is input for processing and analysis. A special form of comment (optional) inserted by the user directs JAVS processing for program performance analysis.

JAVS analyzes the code and generates a directed graph of the control structure. The possible flows through the program are determined. All pertinent data is stored in a data base for later use. Additional or changed source code causes an existing data base to be updated.

JAVS automatically inserts software probes into the source code to intercept and record program flow during execution. A second type of instrumentation is used to record statistics on program performance according to directives inserted by the user in the source.

JAVS provides a variety of services which assist in the establishing of a continuing testing strategy both at the detailed level within a module and at the module interdependence level. Test cases are constructed by the user. Reports can be used as program documentation.

Program execution provides normal computational results as well as outputs from the instrumentation. Structural flow output is recorded for later post-test analysis by JAVS. User-directed program performance output is interspersed with normal program output.

JAVS includes detailed post-test analysis facilities which provide measures of testing thoroughness, both individually and cumulatively for a set of test cases.

The results are examined by the user to determine if test goals have been met and testing is completed.

AN-47751

Figure 4.1. The Role of Documentation in the Testing Process

- Obtain the three module documentation reports for selected modules of a specified JAVSTEXT, in addition to the four library-wide reports.

```
DOCUMENT,JAVSTEXT = <textname>,
      MODULE = <name-1>,...,<name-n>.
```

- Specify a library name (must be the same as the created library name), and select the DD-path picture and definition reports for all modules, in addition to the seven macro command reports.

```
OLD LIBRARY = <libname>.

START.

DOCUMENT.

FOR LIBRARY.

PRINT,DDPATHS                        (Fig. 4.9)

ASSIST,PICTURE.                      (Fig. 4.10)

END FOR.
```

4.3 DOCUMENTATION OUTPUT

Figures 4.2-4.8 contain sample output for the seven reports generated by the DOCUMENT macro command. The first four reports are library-wide reports; the last three reports (Figs. 4.6-4.8) are module reports. Figures 4.9 and 4.10 show sample output for the additional commands:

```
PRINT,DDPATHS.

ASSIST,PICTURE.
```

```
GENERAL CROSS REFERENCE LISTING

    FOR WHOLE LIBRARY

SYMBOL     MODULE     USED/SET/DEFINITION ( * INDICATES SET, D INDICATES DEFINITION )

BG         EXPROGM      18D   28
CARD       EXPROGM      12D   18*   21    22    23
EXMPL1     EXMPL1        1
           EXPROGM      24
EXMPL2     EXMPL2        1
           EXPROGM      27
EXMPL3     EXMPL1       29
           EXMPL3        1
FILL       EXMPL1        5D   15*
ID         EXPROGM       5D   15    21
INDXS      EXMPL1        7D   22*   24*   28
ITER1A     EXPROGM       8D   10    15    22
ITER1      EXMPL2        4*
           EXPROGM       6D   10D   24    26
ITER2A     EXPROGM       9D   11    15    23
ITER2      EXMPL2        5*
           EXPROGM       7D   11D   24
LABEL1     EXMPL1       27    27    27    30D
LABEL2     EXMPL1       27    29D
LIMIT1     EXMPL1        1     3D    9    10*   12
LIMIT2     EXMPL1        1     4D   18
MESAGE     EXCOMPL       5D
MESSAG     EXCOMPL      11D
           EXMPL1       31*   33*   35*   38
           EXPROGM      16*   17
MSG1       EXCOMPL       7D
           EXPROGM      16
MSG2       EXCOMPL       8D
           EXMPL1       31
MSG3       EXCOMPL       9D
           EXMPL1       33
MSG4       EXCOMPL      10D
           EXMPL1       35
PICK       EXMPL1       27D   28
PRINTR     EXMPL1       38
           EXPROGM      14D   17
READER     EXPROGM      13D   18    19
RESULT     EXMPL1        6D   11*   16*   16    30    32
           EXMPL3        4*
TAPE5      EXPROGM      13
TAPE6      EXPROGM      14
```

This report provides a symbol cross reference listing for all modules on the library. The symbol types are variables, file names, switch names, labels, and subprogram names. Adjacent to the statement number of the symbols appearance is a flag (* or D) indicating setting or definition.

Figure 4.2. Library-Wide Symbol Cross Reference

```
LIBRARY DEPENDENCE TABLE
•---------•------------------------•
•.I       •                        •
•I.N      •AABCCCDEEGGGLLMMNOOPPSST•
• N.V     •DDTIKOTFREETOUUUPLURVBGE•
•  V.O    •CDOUBNOARTTCOKVVUOTMAPNR•
•   O.K   •RTDNKTBCXBCRK 14TOS LTPM•
•    K.E  • O T O  R        KB UR I•
•     E.E • K   K                 N•
•      R. •                        •
•---------•------------------------•
• ADCR    •.       X     XX        •
• ADDTOK  • .   X  X         XX    •
• BTOD    •  .                     •
• CIDNT   •   .                    •
• CKBK    •    .      X            •
• CONTOK  •     .                  •
• DTOB    •      .                 •
• EFAC    •  X    .      XX        •
• ERRXR   •  X     .               •
• GETB    •         .              •
• GETC    •          .             •
• GTCR    •        X  .    X      X•
• LOOK    • X      X   .    XXX    •
• LUK     •   X    X XX .X     X   •
• MUV1    •        X     .         •
• MUV4    •         X    X.        •
• NPUT    •        X       .      X•
• OLOOK   •X XXX XXXX X XXXX.  X XX•
• OUTSB   •                  .     •
• PRM     •                   . X  •
• PVALU   •        X           .   •
• SBPTR   •        X            .  •
• SGNP    •        X    XX       . •
• TERMIN  •  X     X              .•
•---------•------------------------•
```

The interaction of all modules on the data base library is shown in this square matrix. If the library contains all modules in the user's program, this report provides a concise, complete picture of the total internal module dependencies. If the library contains a subset of the total program, this report aids in determining what modules do not interact with the component and might be better suited for another component. The modules are listed in alphabetical order.

Figure 4.3. Internal Library Dependencies

```
AUXILIARY LIBRARY DEPENDENCE TABLE
◆---------◆-----------------------◆
◆.I       ◆                       ◆
◆I.N      ◆ABEFGGGHIIMMMMOPPPRRSSW◆
◆ N.V     ◆CTRAETVONNDDDUPPUUETDYR◆
◆  V.O    ◆SORTTCELMSBBBVUXTTMSBSA◆
◆   O.K   ◆THOABAUSDBNNS2TXBLQ STP◆
◆    K.E  ◆B RLLRP B AET   LSU TEU◆
◆      E.E◆    KD    MX    KTO RMP◆
◆       R.◆                     P ◆
◆---------◆-----------------------◆
◆ ADCR    ◆                       ◆
◆ ADDTOK  ◆       X          XX   ◆
◆ BTOD    ◆                       ◆
◆ CIDNT   ◆                  X    ◆
◆ CKRK    ◆                       ◆
◆ CONTOK  ◆       X          X    ◆
◆ DTOB    ◆                       ◆
◆ EFAC    ◆                       ◆
◆ ERRXR   ◆  X           X      X ◆
◆ GETB    ◆                  X    ◆
◆ GETC    ◆                  X    ◆
◆ GTCR    ◆                       ◆
◆ LOOK    ◆       X  X            ◆
◆ LUK     ◆             X         ◆
◆ MUVI    ◆                       ◆
◆ MUV4    ◆                       ◆
◆ NPUT    ◆ X   X XX      X       ◆
◆ OLOOK   ◆                       ◆
◆ OUTSB   ◆X   X    X XX   XX  X  ◆
◆ PRM     ◆                       ◆
◆ PVALU   ◆      X           X    ◆
◆ SBPTR   ◆                       ◆
◆ SGNP    ◆                       ◆
◆ TERMIN  ◆   X          X       X◆
◆---------◆-----------------------◆
```

This report differs from the last one in that the invoked modules (listed across the page) are not resident on the library. If the library allegedly contains all modules in the program, the external modules should consist only of system routines. If the library contains a component of the total program, this report shows the module invocation interfaces to other externals.

Figure 4.4. External Library Dependencies

```
LIBRARY DEPENDENCE SUMMARY...

THE FOLLOWING PROCEDURES ARE NOT INVOKED BY ANY MODULE ON THE LIBRARY

        LOOK

THE FOLLOWING PROCEDURES DO NOT INVOKE ANY MODULE ON THE LIBRARY
        ( * PROCEDURES DO NOT INVOKE ANY MODULES AT ALL )

        CONTOK
        OUTSB
        GETB
        GETC
        BTOD     *
        CIDNT
        DTOB     *
```

Considering the modules on the library as a pyramid representing the invocation hierarchy of the modules, this report identifies the "top" and "bottom" modules in the system.

Figure 4.5. Library High- and Low-Level Modules

```
MODULE STATEMENT LISTING

MODULE <EXMPL1  >. JAVSTEXT <EXPROGM >. PARENT MODULE <EXPROGM >

 NO.  LVL              STATEMENT                                               DD-PATHS        CONTROL
---------------------------------------------------------------------------------------------------------
  1 ( 0)               PROC EXMPL1 ( LIMIT1 , LIMIT2 ) $                       (   1)
  2 ( 1)                 BEGIN
  3 ( 1)                 ITEM LIMIT1 I 24 S $
  4 ( 1)                 ITEM LIMIT2 I 24 S $
  5 ( 1)                 ARRAY FILL 100 I 24 S $
  6 ( 1)                 ITEM RESULT I 24 S $
  7 ( 1)                 ITEM INDXS I 24 S $
  8 ( 1)               ##. TRACE , RESULT ##
  9 ( 1)                 IF LIMIT1 GQ 100 $                                    (   2-   3)    IF
 10 ( 2)                   LIMIT1 = 99 $
 11 ( 1)                 RESULT = 4 $
 12 ( 1)                 FOR I = 1 , 1 , LIMIT1 $                                              FOR3
 13 ( 2)                   BEGIN
 14 ( 2)               ##. EXPECT , RESULT = 1 , 5 ##
 15 ( 2)                   FILL ($ I - 1 $) = I $
 16 ( 2)                   RESULT = ( RESULT + I ) / I $
 17 ( 2)               ##. ASSERT , RESULT GR 10 ##
 18 ( 2)                   FOR J = 1 , 1 , LIMIT2 $                                            FOR3
 19 ( 3)                     BEGIN
 20 ( 3)               ## CLOSE ##
 21 ( 3)                     IFEITH J LQ 3 $                                   (   4-   5)    IFEI
 22 ( 4)                       INDXS = J $
 23 ( 3)                     ORIF 1 $                                          (   6)         ORIF
 24 ( 4)                       INDXS = 4 $
 25 ( 3)                     END
 26 ( 3)               ## IFEITH ##
 27 ( 3)                     SWITCH PICK = ( LABEL1 , LABEL1 , LABEL1 , LABEL2 ) $
 28 ( 3)                     GOTO PICK ($ INDXS - 1 $) $                       (   7-  11)         ----->
 29 ( 3)          LABEL2.                                                                    INV <-----
                             GOTO EXMPL3 $
 30 ( 3)          LABEL1.                                                      (  12-  13)    IFEI<-----
                             IFEITH RESULT LS 4 $
 31 ( 4)                       MESSAG = MSG2 $
 32 ( 3)                     ORIF RESULT EQ 4 $                                (  14-  15)    ORIF
 33 ( 4)                       MESSAG = MSG3 $
 34 ( 3)                     ORIF 1 $                                          (  16)         ORIF
 35 ( 4)                       MESSAG = MSG4 $
 36 ( 3)                     END
 37 ( 3)               ## IFEITH ##
 38 ( 3)                     OUTPUT PRINTR MESSAG $                                            I/O
 39 ( 3)                     END                                               (  17-  18)
 40 ( 2)               ## J ##
 41 ( 2)                   END                                                 (  19-  20)
 42 ( 1)               ## I ##
 43 ( 1)               ##. OFFTRACE , RESULT ##
 44 ( 1)                 END
---------------------------------------------------------------------------------------------------------
```

This module report is a listing of the source statements enhanced by the control nesting level, indentation of each control level, decision points flagged with their DD-path numbers, control statement abbreviations, and arrows showing potential control transfers within and out of the module.

Figure 4.6. Module Listing

```
MODULE INVOCATION BANDS

MODULE <NPUT    >, JAVSTEXT <NPUT    >, PARENT MODULE <NPUT    >

LEVEL                               -2       -1         0         1         2
-----------------------------------------------------------------------------------------------
                                                      NPUT
                                            GTCR                BTOH
                                  CKBK                          ERRXR
                                  LUK                                     BTOD
                                  OLOOK                                   ERROR
                                            OLOOK                         OPUT
                                  LOOK                                    SYSTEMP
                                                                GTCARD

                                                                PRXX
                                                                TERMIN
                                                                          BTOD
                                                                          ERRXR
                                                                          FATAL
                                                                          OPUT
                                                                          WRAPUP
-----------------------------------------------------------------------------------------------
```

This report shows the selected module within the invocation hierarchy. At the center is the specified module. Each successive band of modules from the center to the left shows the calling modules; each successive band to the right shows the called modules. The left (calling) modules reside on the library; the right (called) modules can include modules external to the JAVS library. Five bands on each side of the specified module are displayed when the DOCUMENT macro command is used. The band width is a user option. Within each band, the modules are listed alphabetically.

Figure 4.7. Module Invocation Bands

```
MODULE INVOCATION SPACE...
MODULE <NPUT    >, JAVSTEXT <NPUT    >, PARENT MODULE <NPUT    >
------------------------------------------------------------------------
   3            PROC NPUT $
------------------------------------------------------------------------
INVOCATIONS FROM WITHIN THIS MODULE
----------------------------------
MODULE RTOH
                  STMT =   68        BTOH ( LYNCNT )
                  STMT =   69        BTOH ( STCTR )
MODULE ERRXR
                  STMT =   29        ERRXR ( 4 )
MODULE GTCARD
                  STMT =   41        GTCARD ( = STATUS , CRD1 )
MODULE PRXX
                  STMT =   79        PRXX ( HEL , 120 , EJCT )
MODULE TERMIN
                  STMT =   49        TERMIN ( 1 )
------------------------------------------------------------------------
INVOCATIONS TO THIS MODULE FROM WITHIN LIBRARY
--------------------------------------------
MODULE GTCR
                  STMT =   31        NPUT
MODULE OLOOK
                  STMT =  119        NPUT
                  STMT =  306        NPUT
                  STMT =  363        NPUT
------------------------------------------------------------------------
```

This module report shows all invocations, along with the statement numbers, to and from the specified module. It is useful in examining actual parameter usage.

Figure 4.8. Module Invocation Space

```
MODULE DD-PATH DEFINITION LISTING

MODULE <EXMPL1   >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

 NO.  LVL              STATEMENT                                          DD-PATHS GENERATED
------------------------------------------------------------------------------------------------------------
  1 ( 0)               PROC EXMPL1 ( LIMIT1 , LIMIT2 ) $
                                                                    ** DD-PATH    1 IS PROCEDURE ENTRY
                         . . . .
  9 ( 1)                 IF LIMIT1 GQ 100 $
                                                                    ** DD-PATH    2 IS TRUE BRANCH
                                                                    ** DD-PATH    3 IS FALSE BRANCH
                         . . . .
 12 ( 1)                 FOR I = 1 , 1 , LIMIT1 $
 13 ( 2)                   BEGIN
                           . . . .
 18 ( 2)                   FOR J = 1 , 1 , LIMIT2 $
 19 ( 3)                     BEGIN
                             . . . .
 21 ( 3)                     IFEITH J LQ 3 $
                                                                    ** DD-PATH    4 IS TRUE BRANCH
                                                                    ** DD-PATH    5 IS FALSE BRANCH
                             . . . .
 23 ( 3)                     ORIF 1 $
                                                                    ** DD-PATH    6 IS TRUE BRANCH
                             . . . .
 25 ( 3)                     END
                             . . . .
 27 ( 3)                     SWITCH PICK = ( LABEL1 , LABEL1 , LABEL1 , LABEL2 ) $
 28 ( 3)                     GOTO PICK ($ INDXS - 1 $) $
                                                                    ** DD-PATH    7 IS SWITCH OUTWAY     1
                                                                    ** DD-PATH    8 IS SWITCH OUTWAY     2
                                                                    ** DD-PATH    9 IS SWITCH OUTWAY     3
                                                                    ** DD-PATH   10 IS SWITCH OUTWAY     4
                                                                    ** DD-PATH   11 IS SWITCH OUTWAY     5
                             . . . .
 30 ( 3)           LABEL1.
                             IFEITH RESULT LS 4 $
                                                                    ** DD-PATH   12 IS TRUE BRANCH
                                                                    ** DD-PATH   13 IS FALSE BRANCH
                             . . . .
 32 ( 3)                     ORIF RESULT EQ 4 $
                                                                    ** DD-PATH   14 IS TRUE BRANCH
                                                                    ** DD-PATH   15 IS FALSE BRANCH
                             . . . .
 34 ( 3)                     ORIF 1 $
                                                                    ** DD-PATH   16 IS TRUE BRANCH
                             . . . .
 36 ( 3)                     END
                             . . . .
 39 ( 3)                     END
                                                                    ** DD-PATH   17 IS LOOP ON FOR AGAIN
                                                                    ** DD-PATH   18 IS ESCAPE FOR LOOP
                           . . . .
 41 ( 2)                   END
                                                                    ** DD-PATH   19 IS LOOP ON FOR AGAIN
                                                                    ** DD-PATH   20 IS ESCAPE FOR LOOP
------------------------------------------------------------------------------------------------------------
```

This report is useful for documentation purposes because it defines the outways of all decisions and makes the decision points more visible by omitting the intervening sequential statements. The last switch outway is the "drop through" path.

Figure 4.9. DD-Path Definitions

```
PICTURE WITH ALL DD-PATHS...

MODULE <EXAMPL  >, JAVSTEXT <EXAMPL  >, PARENT MODULE <EXAMPL  >

                                                 STMT  STMT
(B = BEGIN, E = END, S = SELF-LOOP)              TYPE  NO.                 DD-PATH NUMBERS...
----------------------------------------------------------------------------------------------
                                                 <PROC   3> B                           1
                                                            ↓
                                                 <IFEI  15> EBB                         2   3
                                                             ↓↓
                                                 <ORIF  17> B↓EB                        4   5
                                                            ↓↓ ↓
                                              ES <ASMT  23> E↓B↓                        6   7
                                              ↓              ↓↓↓
                                              B  <END   24> B↓E↓                        9   8
                                                            ↓↓ ↓
                                                 <ORIF  27> ↓↓BE                       10
                                                            ↓↓↓
                                               S <ASMT  31> ↓EEB                       11  12
                                                            ↓  ↓
                                                 <END   32> E  E
----------------------------------------------------------------------------------------------
```

This report pictorially illustrates the program flow between decisions in the module. Each B → E sequence is a single decision outway; each outway is assigned the unique DD-path number shown at the right of the report. Any control flow which starts and ends on the same statement number (as in the decisions to "loop again" on JOVIAL FOR statements) is marked with an S for a self-loop.

Figure 4.10. Control Flow Picture

5 INSTRUMENTATION

JAVS instrumentation automatically inserts a set of probe statements into each module to capture control and record information during execution. The module can be compiled in instrumented form and used in Test Execution. The probed modules are logically equivalent to the original source. Instrumentation can be performed before or after the documentation reports have been obtained. Figure 5.1 shows the role of instrumentation in the context of the testing process.

5.1 TASKS

There are two types of instrumentation:

1. Structural Instrumentation. Software probes are inserted into the source text at each invocation point, each return point, and each statement which begins a DD-path. Each probe includes a call to special data collection routines which capture and record information concerning flow of control in the executing module(s).

2. JAVS Directive Instrumentation. Software probes which monitor the results of assignment and exchange statements during Test Execution are inserted in the source text where the user has placed JAVS computation directives (Sec. 1.5 of the Reference Manual). Each directive controls execution-time output which is interspersed with normal program output.

In addition to the structural probes, software probes are inserted to indicate a new test case and to indicate the end of the test. The instrumented source is written onto the LPUNCH file, where it can be input to the JOVIAL compiler. During execution of the instrumented program (Test Execution), the data collection probes record on an audit file which is analyzed by JAVS to provide execution tracing and coverage reports.

5.2 INSTRUMENTATION INPUT

JAVS instrumentation requires the data base containing syntax and structural analyses for all modules being instrumented plus a set of commands to direct the instrumentation.

5.3 COMMANDS

The macro command

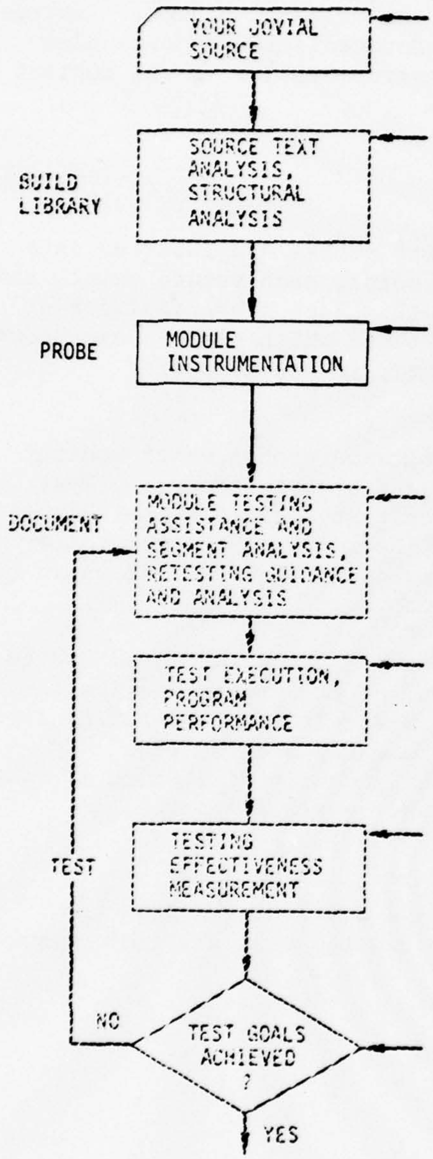
(1) PROBE,JAVSTEXT = <text name>.

causes all modules in the named JAVSTEXT (START-TERM sequence) to be instrumented and written to file LPUNCH for compilation. The above macro command expands into the following standard commands:

(2) OLD LIBRARY = TEST.

START.

AN-47751



JOVIAL source code is input for processing and analysis. A special form of comment (optional) inserted by the user directs JAVS processing for program performance analysis.

JAVS analyzes the code and generates a directed graph of the control structure. The possible flows through the program are determined. All pertinent data is stored in a data base for later use. Additional or changed source code causes an existing data base to be updated.

JAVS automatically inserts software probes into the source code to intercept and record program flow during execution. A second type of instrumentation is used to record statistics on program performance according to directives inserted by the user in the source.

JAVS provides a variety of services which assist in the establishing of a continuing testing strategy both at the detailed level within a module and at the module interdependence level. Test cases are constructed by the user. Reports can be used as program documentation.

Program execution provides normal computational results as well as outputs from the instrumentation. Structural flow output is recorded for later post-test analysis by JAVS. User-directed program performance output is interspersed with normal program output.

JAVS includes detailed post-test analysis facilities which provide measures of testing thoroughness, both individually and cumulatively for a set of test cases.

The results are examined by the user to determine if test goals have been met and testing is completed.

Figure 5.1. The Role of Instrumentation in the Testing Process

```
JAVSTEXT = <text name>.
FOR JAVSTEXT.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = <text name>,INSTRUMENTED = ALL.
END.
```

Since the JOCIT JOVIAL compiler at RADC accepts only one START-TERM sequence in a single compilation activity, it is recommended that the user instrument only one JAVSTEXT per JAVS run; i.e., use only one PROBE macro command.

Prior to execution of the instrumented modules, the user must specify the test initialization and termination by invoking one of the JAVS data collection routines. The insertion of these invocation statements can be performed manually or automatically by using the following commands during instrumentation:

```
(3)    OLD LIBRARY = <libname>.
       START.
       PROBI,STARTTEST = <module name>,<text name>,
               <statement no.>,<test name>,<tracing level>.
       PROBI,STOPTEST = <module name>,<text name>,
               <statement no.>.
       PROBE,JAVSTEXT = <text name>.
       PRINT,JAVSTEXT = <text name>, INSTRUMENTED = ALL.
```

All of the command rules associated with this set of instrumentation commands are described in Appendix B. A description of the PROBI command and its options is in Sec. B.2.4.

5.4 INSTRUMENTATION OUTPUT

The primary output is the instrumented source code which is normally written to the LPUNCH file [note the PUNCH command in the expansion of the PROBE macro in Sec. 5.3 (2)] during the instrumentation process. At user request, the instrumented source code can be saved on the library to be written to LPUNCH at a later time.

Printed output from instrumentation consists of a short description of each probed module to inform the user of the extent of instrumentation performed (see Sec. 4.4.3 of the Reference Manual for instrumentation options). The user can request that the probed text be printed at the end of the instrumentation activity, or at any later time if the probed code was saved on the library, by using the PRINT command in Sec. 5.3 (3).

6 TEST EXECUTION

Once the JOVIAL source code has been instrumented by JAVS, either before or after obtaining the documentation reports, the instrumented code can be executed with test data. Figure 6.1 shows the Test Execution process in the context of the entire testing process.

The instrumented source code output on file LPUNCH is compiled with a COMPOOL which supplies definitions for the JAVS data collection procedures. The compiled code is then loaded with the JAVS data collection procedures from the JAVS object code library and any other externals which are necessary. During Test Execution the program operates normally, reading its own data and writing its own outputs. The instrumented modules call the data collection routines which record on the test file, AUDIT, an execution trace and accumulated data on module invocations and DD-path traversals. Performance data resulting from JAVS computation directives are interspersed with normal program output to the printer. (See Sec. 1.5 in the Reference Manual.) The data base library is not used during Test Execution.

Each Test Execution may consist of a number of test cases. The program identifies the start of each new test case by executing a call to one of the data collection routines (PROBI); the end of all test cases is similarly treated. These identification calls are automatically inserted by using the PROBI,STARTTEST and PROBI,STOPTEST commands during instrumentation or are manually inserted by the user in the program prior to compilation; all other instrumentation of the source is performed automatically.

6.1 TASKS

Test Execution differs from normal execution of the program in four respects:

1. Some or all of the program has been instrumented.
2. Test case boundaries are identified.
3. The instrumented code is compiled with a JAVS COMPOOL.
4. Data collection routines are added to the load sequence.

Figure 6.2 shows a diagram of the Test Execution process.

6.2 REFERENCES TO DATA COLLECTION ROUTINES AND TEST FILE

The instrumented source contains invocations of the data collection procedures which were not in the original user source. If the user's program has a COMPOOL, the procedure declarations for the JAVS data collection routine (PROBM, PROBE, PROBI, and PROBD) must be added to the COMPOOL. If the program has no COMPOOL, the one shown in Fig. 6.3 should be used.* This is necessary in order to compile the instrumented source without compilation errors.

---

*If the JOVIAL compiler accepts a list of processed COMPOOLs while compiling executable modules, then the one shown in Fig. 6.3 must be included. The JOCIT compiler requires the FILE declaration to be in the COMPOOL.

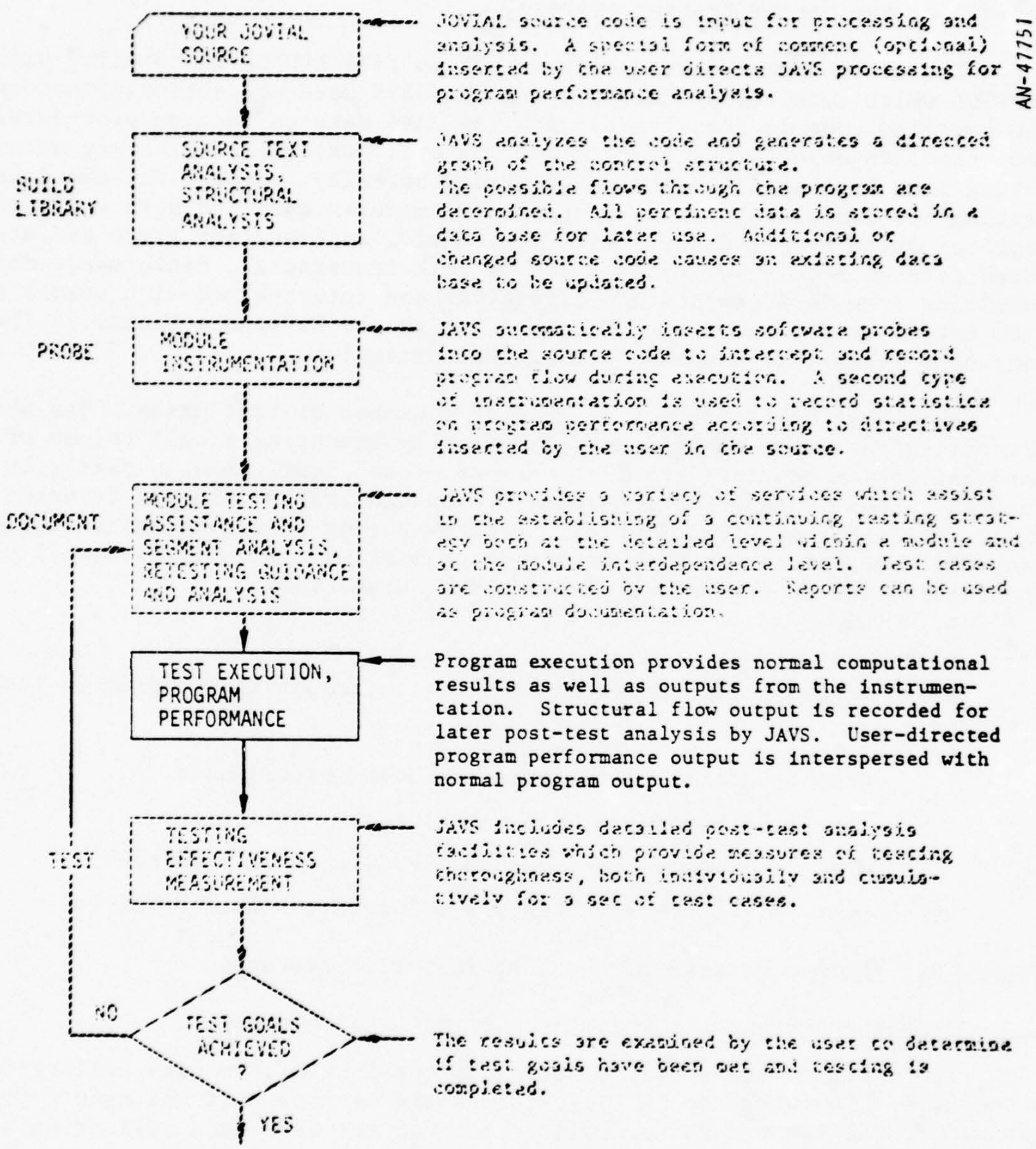
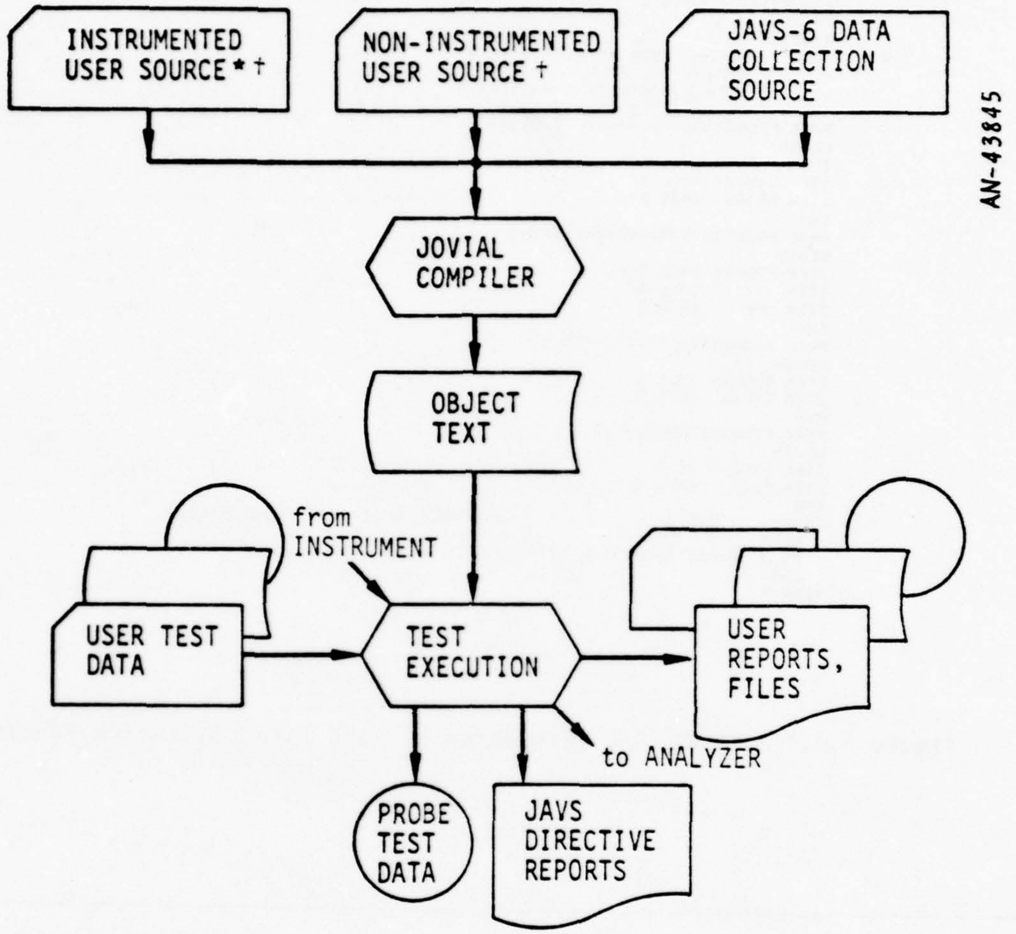


Figure 6.1. The Role of Test Execution in the Testing Process



*FROM INSTRUMENT

†MAY ALSO CONTAIN CALLS TO PROBI AND PROBD

Figure 6.2. Test Execution Processing

```
**.JAVSTEXT PRCMPL PRESET = COMPOOL FOR PROBE ANALYSIS DURING EXECUTION
**
START $
        DEFINE INTG ** I 24 S ** $
        DEFINE HLL ** H 4 ** $
        DEFINE DINTG ** I 48 S ** $
        DEFINE DHLL ** H 8 ** $
PROC PROBM(MODNAM,JAVTXT,NODPS)$
BEGIN
ITEM MODNAM DHLL $
ITEM JAVTXT DHLL $
ITEM NODPS  INTG $
END
PROC PROBE(MODNAM,JAVTXT,DDP)$
BEGIN
ITEM MODNAM DHLL $
ITEM JAVTXT DHLL $
ITEM DDP    INTG $
END
PROC PROBI(TESNAM,TFLAG)$
BEGIN
ITEM TESNAM DHLL $
ITEM TFLAG  INTG $
END
PROC PROBD(LINE,FLAG)$
BEGIN
ITEM LINE H 80 $
ITEM FLAG   INTG $
END
COMMON PROBEF $                  **COMMON BLOCK FOR PROBE FILE**
  BEGIN
  FILE AUDIT B 84 R 0 V(OK) V(X1) V(X2) V(X3) V(EOF) 08$
  END
TERM $
```

Figure 6.3. COMPOOL for References to JAVS Data Collection Routines

A FILE declaration for the test file AUDIT (08) must also appear. The location of the FILE declaration (i.e., in a COMPOOL or in the main program) and its format are dependent on the JOVIAL compiler being used. The example in Fig. 6.3 is for RADC.

Job streams for compiling the instrumented source code, loading, and executing are given in Appendix D. Test Execution operates without the JAVS environment, except for the data collection routines. Thus, there are no JAVS commands, data base library or JAVS reports associated with this process.

6.3 TEST CASE IDENTIFICATION AND TEST FILE CONTROL

At appropriate places in the instrumented source program (i.e., where a new test case begins and at the end of all test cases) a call to PROBI must be inserted. PROBI performs two services: it identifies each test case and controls the recording of data on the test file. PROBI has two parameters: the first is used as a test case name on ANALYZER reports and is a Hollerith name of eight characters; the second is used to control the amount of data actually recorded on the test file.

The possible values for the test file control parameter TFLAG are shown in Table 6.1. A Zero value signals the end of all test cases. A non-zero value signals the start of a new test case (and the end of the previous test case, if any). The value of the second parameter (if nonzero) determines the amount of execution tracing. If TFLAG is 1, no tracing is maintained; if TFLAG is 2, invocations and returns are traced; if TFLAG is 3, invocations, returns, and DD-paths are traced.

TABLE 6.1

TEST FILE DATA CONTROL WITH PROBI

TFLAG	SIGNAL	TEST-FILE DATA RECORDED	ANALYZER REPORT OPTIONS
0	end-of-test file	(last) test case summary	
1	new test case	test-case summary	SUMMARY, HIT, NOTHIT, MODLST, DDPATHS
2	new test case	test-case summary, module invocation/return trace	SUMMARY, HIT, NOTHIT, MODLST, TIME, MODTRACE, DDPATHS
3	new test case	test-case summary, module invocation/return trace, DD-path execution trace	SUMMARY, HIT, NOTHIT, MODLST, TIME, MODTRACE, DDPTRACE, DDPATHS

7 POST-TEST ANALYSIS

The Test Effectiveness Measurement Analyzer provides a detailed and comprehensive analysis of testing coverage. ANALYZER (JAVS standard command keyword for this functional process) generates reports on execution tracing, coverage, timing and paths not taken. The ANALYZER Process is the end of one revolution in the automated testing cycle. Armed with the JAVS reports showing the program's execution behavior, the tester can determine whether further testing is necessary. Figure 7.1 shows where the post-test analysis fits into the testing process.

7.1 TASKS

ANALYZER reads the AUDIT file which was generated and saved during Test Execution. Structural data are input from the data base library, and various reports are produced at user request which show the extent of program coverage provided by the test cases.

7.2 ANALYZER INPUT

The data base library, containing syntax and structural analyses, along with the execution trace (AUDIT) file and JAVS commands are input for post-test analysis.

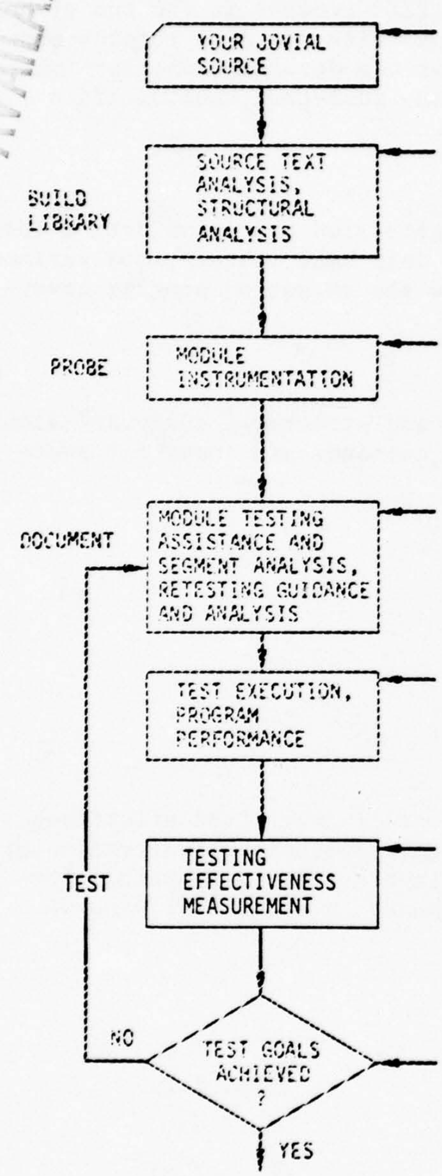
7.3 COMMANDS

The JAVS command set

```
OLD LIBRARY = <libname>.
START.
ANALYZER,MODLST.
TEST.
```

provides a collection of reports useful as the preliminary test effectiveness measurement. These reports contain statement coverage, execution tracing of modules, DD-paths not taken, and a summary of invocation and DD-path information by test case. The above command set expands into the following JAVS standard commands:

```
OLD LIBRARY = <libname>.
START.
ANALYZER,MODLST.                              (Fig. 7.2)
ANALYZER,ALL MODULES.
ANALYZER,SUMMARY.                             (Fig. 7.3)
ANALYZER,NOTHIT.                              (Fig. 7.4)
ANALYZER,MODTRACE.                            (Fig. 7.5)
ANALYZER.
END.
```



AN-47751

JOVIAL source code is input for processing and analysis. A special form of comment (optional) inserted by the user directs JAVS processing for program performance analysis.

JAVS analyzes the code and generates a directed graph of the control structure. The possible flows through the program are determined. All pertinent data is stored in a data base for later use. Additional or changed source code causes an existing data base to be updated.

JAVS automatically inserts software probes into the source code to intercept and record program flow during execution. A second type of instrumentation is used to record statistics on program performance according to directives inserted by the user in the source.

JAVS provides a variety of services which assist in the establishing of a continuing testing strategy both at the detailed level within a module and at the module interdependence level. Test cases are constructed by the user. Reports can be used as program documentation.

Program execution provides normal computational results as well as outputs from the instrumentation. Structural flow output is recorded for later post-test analysis by JAVS. User-directed program performance output is interspersed with normal program output.

JAVS includes detailed post-test analysis facilities which provide measures of testing thoroughness, both individually and cumulatively for a set of test cases.

The results are examined by the user to determine if test goals have been met and testing is completed.

Figure 7.1. The Role of Post-Test Analysis in the Testing Process

If the testing goal is execution of all (or most) decision outways (DD-paths), then the command

ANALYZER,DDPATHS. (Fig. 7.6)

should be specified in addition to or in lieu of ANALYZER,MODLST.

7.4 ANALYZER OUTPUT

The reports produced during post-test analysis for the sample command set in Sec. 7.4 are shown in Fig. 7.2 through 7.6. Additional reports containing module execution timing, DD-path tracing, and DD-paths executed for each test case are described in the Reference Manual.

In the report descriptions, the "specified module" is determined by the user. The TEST macro command used in the sample command set contains the module specification: ANALYZER,ALL MODULES which refers to all modules on the data base library. The user can specify a subset of modules by using the macro command

TEST,MODULE = <name-1>,...,<name-n>.

or the standard command

ANALYZER,MODULE = <name-1>,...,<name-n>.

```
MODULE STATEMENT LISTING

MODULE <EXPROGM >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

 NO.  LVL              STATEMENT                                                 DD-PATHS              CONTROL
-----------------------------------------------------------------------------------------------------------------
   1 ( 0)              ##, JAVSTEXT EXPROGM COMPUTE ( EXCOMPL ) ##
   2 ( 0)              START
   3 ( 0)              $
   4 ( 0)              ## JOVIAL SIMPLE TEST PROGRAM ##
   5 ( 0)              ITEM ID H 4 $
   6 ( 0)              ITEM ITER1 I 24 S $
   7 ( 0)              ITEM ITER2 I 24 S $
   8 ( 0)              ITEM ITER1A H 4 $
   9 ( 0)              ITEM ITER2A H 4 $
  10 ( 0)              OVERLAY ITER1 = ITER1A $
  11 ( 0)              OVERLAY ITER2 = ITER2A $
  12 ( 0)              ITEM CARD H 80 $
  13 ( 0)              FILE READER H 0 R 84 V(OK) V(EOF) TAPE5 $
  14 ( 0)              FILE PRINTR H 0 R 128 V(OK) V(EOF) TAPE6 $
  15 ( 0)              MONITOR ID , ITER1A , ITER2A $                                        0 *
  16 ( 0)              MESSAG = MSG1 $                                   (  1)               1
  17 ( 0)              OUTPUT PRINTR MESSAG $                                                1    I/O
  18 ( 0)          BG.                                                                       3    I/O <-----
                       INPUT READER CARD $
  19 ( 0)              IF READER NO V(EOF) $                             (  2-   3)          3    IF
  20 ( 1)                BEGIN                                                               2
  21 ( 1)                BYTE ($ 0 , 4 $) ( ID ) = BYTE ($ 0 , 4 $) ( CARD ) $               2
  22 ( 1)                BYTE ($ 0 , 4 $) ( ITER1A ) = BYTE ($ 9 , 4 $) ( CARD ) $           2
  23 ( 1)                BYTE ($ 0 , 4 $) ( ITER2A ) = BYTE ($ 19 , 4 $) ( CARD ) $          2
  24 ( 1)                EXMPL1 ( ITER1 , ITER2 ) $                                          2    INV
  25 ( 1)              ## EXMPL2 ##
  26 ( 1)                IF ITER1 GQ 100 $                               (  4-   5)          2    IF
  27 ( 2)                  GOTO EXMPL2 $                                                     0 *  INV
  28 ( 1)                GOTO BG $                                                           2         ----->
  29 ( 1)                END                                                                 0 *
  30 ( 0)              ## IF ##
  31 ( 0)              STOP $                                                                1
  32 ( 0)              ## EXMPL1 ##
  33 ( 0)              TERM $
-----------------------------------------------------------------------------------------------------------------

EXECUTABLE STATEMENTS       14
STATEMENTS EXECUTED         12
  PER CENT EXECUTED         85.
```

This report, output for each specified module which is invoked during Test Execution, contains the enhanced module listing with cumulative statement coverage values appended. These values, to the right of the DD-path numbers, reflect the number of times each executable statement was executed during all of the test cases in the run. Executable statements not exercised are flagged with an asterisk.

Figure 7.2. Module Statement Listing

JAVS REPORT FOR DD-PATH SUMMARY.   17 CASE(S)   09/29/76

ALL SPECIFIED MODULES ENCOUNTERED ON THE AUDIT FILE   09/29/76

DATA9 1		09/29/76	SUMMARY -- THIS TEST			CUMULATIVE SUMMARY			
MODULE NAME	JAVSTEXT NAME	NUMBER OF DD-PATHS	NUMBER OF INVOCATIONS	DD-PATHS TRAVERSED	PER CENT COVERAGE	NUMBER OF TESTS	NUMBER OF INVOCATIONS	DD-PATHS TRAVERSED	PER CENT COVERAGE
JBPAD	JBPAD	121	1	4	3	1	1	4	3
TCET	JBPAD	7	0	0	0	1	0	0	0
BCIST	JBPAD	7	1	6	85	1	1	6	85
TIST	JBPAD	3	3	2	66	1	3	2	66
JBPID	JBPID	61	0	0	0	1	0	0	0
		• • •		• • •			• • •		
NBPI	JBPID	7	0	0	0	17	1	7	100
BRTCB	JBPID	1	0	0	0	17	85	1	100
CONVER	JBPID	7	0	0	0	17	1154	6	85
***** ALL *****		1164		1	0			486	41

ALL SPECIFIED MODULES NOT INVOKED DURING RUN

MODULE NAME	JAVSTEXT NAME	NUMBER OF DD-PATHS
BCC1	JBPID	5
BIRR	JBPID	1
BIPF	JBPID	1
BIPK	JBPID	1
NECT	JBPID	3
NSCLX	JBPID	1
***** ALL ***** UNINVOKED MODULES		12

SUMMARY FOR ALL SPECIFIED MODULES

NUMBER OF DD-PATHS	DD-PATHS TRAVERSED	PER CENT COVERAGE
1176	486	41

This report summarizes the testing coverage in terms of module invocations and DD-paths traversed. For all specified modules invoked during the test run, the number of invocations and DD-paths exercised for each test case and for the accumulation of test cases is given. If there are any modules specified but not invoked in the test run, they are listed with the number of DD-paths in each module. At the end of the summary report are the cumulative DD-path coverage values.

Figure 7.3. Invocation and DD-path Execution Summary

```
JAVS REPORT FOR DD-PATHS NOT EXECUTED.     4 CASE(S)                   8/04/75   2.53.16

-------------------------------------------------------------------------------------------------------------
MODULE    JAVSTEXT I              TEST                 I  PATHS  I          LIST OF DECISION-TO-DECISION PATHS NOT EXECUTED
 NAME       NAME   I          IDENTIFICATION           I NOT HIT I
-------------------------------------------------------------------------------------------------------------
EXPROGM   EXPROGM  I SAMPLE     1  2209 08/04/75 I     4    I       2   3   4   5
                   I SAMPLE     2  2209 08/04/75 I     3    I       1   3   4
                   I SAMPLE     3  2209 08/04/75 I     3    I       1   3   4
                   I SAMPLE     4  2209 08/04/75 I     4    I       1   2   4   5
                   I     TOTAL NOT HIT           I     1    I       4
-------------------------------------------------------------------------------------------------------------
EXMPL1    EXPROGM  I SAMPLE     1  2209 08/04/75 I   A L L  I
                   I SAMPLE     2  2209 08/04/75 I    11    I       2   5   6   8   9  10  11  12  14  17  19
                   I SAMPLE     3  2209 08/04/75 I    11    I       2   5   6   8   9  10  11  12  14  17  19
                   I SAMPLE     4  2209 08/04/75 I   A L L  I
                   I     TOTAL NOT HIT           I    11    I       2   5   6   8   9  10  11  12  14  17  19
-------------------------------------------------------------------------------------------------------------
```

For all specified modules invoked during the test run, this report lists any DD-path which was not traversed in one or more test cases.

Figure 7.4. DD-Paths not Executed

```
-----------------------------------------------------------------------------------------
                    TEST-CASE                 SAMPLE    1             1049      08/05/75
-----------------------------------------------------------------------------------------
(   0)   EXPROGM (EXPROGM )
-----------------------------------------------------------------------------------------
                    TEST-CASE                 SAMPLE    2             1049      08/05/75
-----------------------------------------------------------------------------------------
(   1)     EXMPL1  (EXPROGM )
-----------------------------------------------------------------------------------------
                    TEST-CASE                 SAMPLE    3             1049      08/05/75
-----------------------------------------------------------------------------------------
(   1)     EXMPL1  (EXPROGM )
-----------------------------------------------------------------------------------------
                    TEST-CASE                 SAMPLE    4             1049      08/05/75
-----------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------
                                TEST-SET TERMINATION
-----------------------------------------------------------------------------------------
```

This report is a printout of the Test Execution trace data at the module invocation and return level. Each line of output represents a module invocation, a new test case (see Sec. 6.3) or the end of the test run. The module and JAVSTEXT names are indented to show invocation.

Figure 7.5. Module Invocation Trace

```
MODULE DD-PATH COVERAGE LISTING

MODULE <EXAMPL1  >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

 NO.  LVL               STATEMENT                                     DD-PATHS GENERATED                    COVERAGE
-------------------------------------------------------------------------------------------------------------------
  1 ( 0)            PROC EXMPL1 ( LIMIT1 , LIMIT2 ) $
                                                              ** DD-PATH   1 IS PROCEDURE ENTRY            2

  9 ( 1)              IF LIMIT1 GO 100 $
                                                              ** DD-PATH   2 IS TRUE BRANCH                0 *
                                                              ** DD-PATH   3 IS FALSE BRANCH               2

 12 ( 1)              FOR I = 1 , 1 , LIMIT1 $
 13 ( 2)                BEGIN
 18 ( 2)                FOR J = 1 , 1 , LIMIT2 $
 19 ( 3)                  BEGIN
 21 ( 3)                  IFEITH J LQ 3 $
                                                              ** DD-PATH   4 IS TRUE BRANCH                2
                                                              ** DD-PATH   5 IS FALSE BRANCH               0 *

 23 ( 3)                  ORIF 1 $
                                                              ** DD-PATH   6 IS TRUE BRANCH                0 *

 25 ( 3)                  END
 27 ( 3)                  SWITCH PICK = ( LABEL1 , LABEL1 , LABEL1 , LABEL2 ) $
 28 ( 3)                  GOTO PICK ($ INDXS - 1 $) $
                                                              ** DD-PATH   7 IS SWITCH OUTWAY     1        2
                                                              ** DD-PATH   8 IS SWITCH OUTWAY     2        0 *
                                                              ** DD-PATH   9 IS SWITCH OUTWAY     3        0 *
                                                              ** DD-PATH  10 IS SWITCH OUTWAY     4        0 *
                                                              ** DD-PATH  11 IS SWITCH OUTWAY     5        0 *

 30 ( 3)         LABEL1.
                          IFEITH RESULT LS 4 $
                                                              ** DD-PATH  12 IS TRUE BRANCH                0 *
                                                              ** DD-PATH  13 IS FALSE BRANCH               2

 32 ( 3)                  ORIF RESULT EQ 4 $
                                                              ** DD-PATH  14 IS TRUE BRANCH                0 *
                                                              ** DD-PATH  15 IS FALSE BRANCH               2

 34 ( 3)                  ORIF 1 $
                                                              ** DD-PATH  16 IS TRUE BRANCH                2

 36 ( 3)                  END
 39 ( 3)                  END
                                                              ** DD-PATH  17 IS LOOP ON FOR AGAIN          0 *
                                                              ** DD-PATH  18 IS ESCAPE FOR LOOP            2

 41 ( 2)                END
                                                              ** DD-PATH  19 IS LOOP ON FOR AGAIN          0 *
                                                              ** DD-PATH  20 IS ESCAPE FOR LOOP            2
-------------------------------------------------------------------------------------------------------------------
TOTAL DDPATHS             20
DDPATHS EXECUTED           9
PERCENT EXECUTED          45
```

For all specified modules invoked during the test run, this report shows the statements which begin each decision outway and the number of times the DD-path was executed during all of the test cases.

Figure 7.6. DD-Path Coverage

8 RETESTING ASSISTANCE

Software testing and retesting can be performed at either the single-module or system level. At the single-module level, the retesting target is a set of decision outways which have not been exercised. Retesting at the system level involves identification of target modules and intermodule dependencies. The JAVS reports for system-level testing and retesting were obtained in the DOCUMENT process. Section 10 and the JAVS Methodology Report[2] describe system-level retesting; this section is devoted to single-module retesting.

Given a partially exercised module, the retesting objective is to devise additional test cases which will increase the testing coverage. Two questions arise in addressing the task of single-module retesting: What are the targets for retesting? And, once selected, what assistance is available to exercise the targets?

8.1 RETESTING TARGETS

The DD-path selection criteria should attempt to maximize collateral testing; i.e., exercising more than one unexercised decision outway with each new test case. Several guidelines can be used to aid the selection:

1. In a cluster of unexercised paths, choose an untested DD-path that is on the highest possible control nesting level. (JAVS statement and DD-path coverage reports print the nesting level number.) This selection assures a high degree of collateral testing, since some of the DD-paths leading to and from the target must be executed.

2. A reaching set is a sequence of DD-paths which lead to a specified decision outway. At user request, JAVS determines reaching sets to include or exclude iteration. Choose a DD-path which is at the end of a long reaching set as the target. In addition to collateral testing, it is likely that the resulting test case input will resemble data which corresponds to the functional nature of the program.

3. If a prior test case carries the program near one of the untested DD-paths, it may be more economical to determine how that test case can be modified to execute the unexercised path. JAVS post-test analysis reports (see ANALYZER,HIT and ANALYZER,DDPTRACE in the Reference Manual) show which DD-paths were executed during each separate test case.

4. If the analyses required for a particular DD-path selection are difficult, then choose a path which lies along the lower level portions of its reaching set. This can simplify the analysis problem.

5. Analyze the untested DD-path predicate (conditional formula) in the reaching set for "key" variable names which may lead directly to the input.

6. Choose DD-paths whose predicates evaluate functional boundaries or extreme conditions; exercising these paths frequently uncovers program errors.

Most of these guidelines depend upon the identification of DD-path reaching sets. One of the tasks performed by the JAVS testing assistance processor (ASSIST) is the determination of these sets.

8.2 REACHING SET TASKS

The user inputs the desired path number to be reached, and ASSIST generates the reaching set of paths from the module entry or from a designated starting path to the specified path. For the generated set of paths, the key program statements are printed, including the necessary outcome of any conditional statements which are essential. The user may specify iterative or noniterative reaching sets to be generated.

8.3 REACHING SET INPUT

The data base library containing syntax and structural analyses plus a set of JAVS commands comprise the input to this process.

8.4 COMMANDS

JAVS standard commands must be used to identify the library, JAVSTEXT, module, DD-path target and, optionally, initial DD-path in the reaching set. Multiple module selection and reaching set commands may be used.

```
OLD LIBRARY = <libname>.
START.
JAVSTEXT = <text name>.
MODULE = <modname>.
ASSIST,REACHING SET, <target>,<initial>.
.
.    other REACHING SET commands for the
.    current module or other module selection
.    and REACHING SET commands
.
END.
```

The target and initial values are DD-path numbers. The initial value is optional; absence of the initial value sets the beginning path to 1. The user can request the iterative reaching set by using the command:

```
ASSIST,REACHING SET,<target>,<initial>,ITERATIVE.
```

8.5 REACHING SET OUTPUT

Figure 8.1 shows the report generated by the command:

```
ASSIST,REACHING SET, 10.
```

```
NON-ITERATIVE REACHING SET
   OF DD-PATH  17 FROM DD-PATH   4

MODULE <EXMPL1   >. JAVSTEXT <EXPROGM >. PARENT MODULE <EXPROGM >

------------------------------------------------------------------------------------------------------------
                    . . .
 18 ( 2)                      FOR J = 1 , 1 , LIMIT2 $
 19 ( 3)                        BEGIN
                    . . .
 21 ( 3)                        IFEITH J LQ 3 $                             *ESSENTIAL PREDICATE*  BEGINNING DDP
                                                                         ** DD-PATH   4 IS TRUE BRANCH
 22 ( 4)                          INDXS = J $
                    . . .
 25 ( 3)                        END
                    . . .
 27 ( 3)                        SWITCH PICK = ( LABEL1 , LABEL1 , LABEL1 , LABEL2 ) $
 28 ( 3)                        GOTO PICK ($ INDXS - 1 $) $
                                                                         ** DD-PATH   7 IS SWITCH OUTWAY    1
                                                                         ** DD-PATH   8 IS SWITCH OUTWAY    2
                                                                         ** DD-PATH   9 IS SWITCH OUTWAY    3
                                                                         ** DD-PATH  10 IS SWITCH OUTWAY    4
                                                                         ** DD-PATH  11 IS SWITCH OUTWAY    5
 29 ( 3)          LABEL2.
                                GOTO EXMPL3 $
 30 ( 3)          LABEL1.
                                IFEITH RESULT LS 4 $
                                                                         ** DD-PATH  12 IS TRUE BRANCH
                                                                         ** DD-PATH  13 IS FALSE BRANCH
 31 ( 4)                          MESSAG = MSG2 $
 32 ( 3)                        ORIF RESULT EQ 4 $
                                                                         ** DD-PATH  14 IS TRUE BRANCH
                                                                         ** DD-PATH  15 IS FALSE BRANCH
 33 ( 4)                          MESSAG = MSG3 $
 34 ( 3)                        ORIF 1 $
                                                                         ** DD-PATH  16 IS TRUE BRANCH
 35 ( 4)                          MESSAG = MSG4 $
 36 ( 3)                        END
                    . . .
 38 ( 3)                        OUTPUT PRINTR MESSAG $
 39 ( 3)                        END                                         *ESSENTIAL PREDICATE*  ENDING DDP
                                                                         ** DD-PATH  17 IS LOOP ON FOR AGAIN
------------------------------------------------------------------------------------------------------------
```

The source listing contains only the key statements of the DD-paths that lead to the specified decision outway. The beginning and ending DD-paths are noted. Any outway which must be taken in order to continue in the reaching set is flagged with "essential predicate."

Figure 8.1 Test Case Assistance REACHING SET

8.6 PROCEEDING FROM REACHING SETS

The process of relating paths which are targets for retesting to the input for generating new test cases is highly dependent upon the design of the test program. JAVS shows what code segments have not been exercised by the data and the program paths that lead up to any selected DD-path target. The tester must analyze the predicates in the testing targets for important variables which may be described in program comments or documentation. These variables can be traced throughout the program by using the JAVS cross reference report and the module interdependency and invocation parameter reports (if the variables are passed as parameters).

When the new set of test cases is generated, it can be added to the previously input data or executed alone by the instrumented modules. The results of this Test Execution are then processed by the post-test analyzer to see if the coverage is satisfactory. At this time, the user may see problems in the software which require code changes. The JAVS documentation reports can be used to determine the effects within a single module or within the data base system of modules that the code modification will have. Retesting requirements for a changed module (or modules) include those modules called by the changed module, and those tests which cause the changed module to execute. DEPENDENCE output provides the module interdependencies; ANALYZER output identifies the test cases which cause the execution of the desired modules.

9 COMMAND SUMMARY

The subset of JAVS commands which will enable the first-time user to process his source code are summarized in Table 9.1. The command streams in Table 9.2 show the natural order of processing and a typical selection of commands. Instrumentation, activity 2 in Table 9.2, is shown as a separate activity so that the user can obtain the statement numbers needed by the PROBI commands. Instrumentation can be performed as part of the first activity following the BUILD LIBRARY operation if the user wishes to manually insert (through a text editor) the necessary invocations to the PROBI data collection routine.

The rules for JAVS macro command usage and the default option values are described in Appendix B. For a complete description of all JAVS commands and sample output each command produces, see the Reference Manual.

TABLE 9.1

COMMAND SUMMARY

	TASK	COMMAND	OPTIONS
(1)	Perform syntax and structural analyses create data base library	BUILD LIBRARY	=<library name>.
(2)	Generate reports for documentation	DOCUMENT	,JAVSTEXT = <text name>, MODULE = <name-1>,..., <name-n>.
(3)	Insert test case initiation	PROBI,STARTTEST = <module name>, <text name>, <statement no.>	,<test case name>, <tracing level>.
(4)	Insert test termination	PROBI,STOPTEST = <module name>, <text name>, <statement no.>.	
(5)	Instrumentation	PROBE	,JAVSTEXT = <text name>, MODULE = <name-1>,..., <name-n>.
(6)	Post-test analysis	TEST	,MODULE = <name-1>,..., <name-n>.
(7)	Retesting	ASSIST,REACHING SET, <target>	,<initial>,ITERATIVE, PICTURE

TABLE 9.2

SAMPLE COMMAND SETS

ACTIVITY	COMMANDS
1	BUILD LIBRARY. DOCUMENT.
2	OLD LIBRARY = TEST. START. PROBI,STARTTEST = <module name>, <JAVSTEXT name>,<statement no.>. . . . PROBI,STOPTEST = <module name>, <JAVSTEXT name>,<statement no.>. PROBE,JAVSTEXT = <JAVSTEXT name>.
3	Perform Test Execution
4	OLD LIBRARY = TEST. START. ANALYZER,MODLST. ANALYZER,DDPATHS. TEST.

10 TESTING METHODOLOGY

How testing is to be accomplished is determined by answering very specific questions:

- What is the software test object? (Sec. 10.1)
- What are the available resources (e.g., hardware, support software, personnel, test time period, test tools)? (Sec. 10.2)
- What are the test goals? (Sec. 10.3)
- What procedure will effectively accomplish the goals? (Secs. 10.4 and 10.5)

There is no single general procedure which applies to all testing situations. Each particular testing activity is distinct and should be analyzed to determine a suitable testing procedure. This suggests that the testing process itself consists of three distinct phases: (1) identifying the elements of the test activity, (2) preparing a test plan and (3) carrying out the planned tests. The remainder of this section addresses the problem of practical application of the testing methodology.

10.1 SOFTWARE TEST OBJECT

The software to be tested is called the software test object.

10.1.1 Source Language

JAVS deals with the software test object in source language form. This is the form most suitable for automated testing because it is the form used to create and to modify the software. JAVS assumes that the source text is free from errors of a syntactical nature (i.e., it compiles without syntax-related errors).

10.1.2 Overall Size

JAVS supports testing of both small and large software objects. The software may consist of a single module or it may be hundreds of modules in size. The total source may range from a few statements to tens of thousands. It may consist of one or more compilation units. The limitations of a specific JAVS implementation are restricted by direct access memory size and the size of auxiliary files in the host computer system.

10.1.3 Organization

The software test object may be a complete or an incomplete program; it may be organized into one or more subsystems, or may be a utility package. If the program is incomplete, some additional software must be supplied for Test Execution. The additional software may be either operational software or test software which, when combined with the software test object, results in a complete program. Although the additional software need not be processed by JAVS in source form, the interfaces to the software test object must be clearly and unambigously defined in order to prepare test data. JOVIAL requires formal interface definitions for both data access and module invocation. The source text for these definitions is essential for JAVS-supported testing.

For a very large software system consisting of many hundreds of modules, it is wise to partition the software into test objects of tractable size. Normally, very large software systems are designed as subsystems according to some functional criteria. If the subsystems themselves are each a collection of hierarchically structured, inter-related components or modules, this same partitioning may also be suitable for testing purposes. Miscellaneous collections of low-level utility modules which are invoked throughout the remainder of the system can be grouped together as a separate subsystem.

In partitioning the software test object, some consideration should also be given to the resources required to test the partition. The instrumented software requires more main memory from code expansion of the software instrumentation probes and more computer time due to the overload of executing the probes. A further consideration is the execution-time behavior of the software test object. During test execution it is important to designate important events (e.g., file activity, link loading, major cycles) which separate the test execution into a sequence of individual tests. This permits the tester to extract more detail from the test results. Candidate events include the start of an initialization or termination process, the start or conclusion of a new test case, the change in mode of behavior (e.g., from normal mode to error mode), opening or closing of files, memory link loading, and invocation of other software not being tested.

To properly partition large software test objects, the tester must use documentation supplied with the software for guidance.* The supplied documentation, which may be manually prepared, may not correspond exactly to the software test object, since more often than not this type of documentation does not reflect the current version of the software. JAVS documentation capabilities offer automated assistance in verifying the accuracy of the supplied software documentation. For example, the intermodule dependence reports give a consise picture of the interface of one set of modules to all referenced modules. Trial partitions of the software may be defined by the users, and verified with these reports from JAVS. If the software is too large to be processed by JAVS as a single unit, initial partitioning must be based on supplied documentation.

10.1.4 Suitability for Testing

There are additional software characteristics which affect the suitability of the test object to JAVS-supported testing. These are a result of assumptions made in the JAVS implementation itself. For example, the current JAVS implementation assumes the software test object contains no recursion, has no concurrent paths during execution, and is not time-critical.

Some software design characteristics facilitate JAVS testing. Among these are:

- Highly modular, structured code

---

*See Sec. 10.4 for partitioning criteria according to software structure.

- Direct correspondence of implemented software to functional specifications
- Localization of code controlling important events in software behavior (e.g., new test case, file I/O, link loading)
- Identifiable module inputs and outputs
- Mnemonic symbol names
- Traceability of symbols to input symbols

Existing software may fortuitously possess some, if not all, of these properties. JAVS testing is hampered if the software test object lacks these, or if it contains logically unreachable code, uses borrowed code, or bypasses normal module invocation and return protocol for control transfer.

Software may also be deliberately designed to take advantage of specific JAVS capabilities such as use of imbedded assertion statements for dynamic checking of expected behavior, automated documentation, or program performance with test point identification.

10.2 TEST RESOURCES

The resources available for testing must be identified before an approach to testing can be determined. These include the computer resources, data for executing the software, the members of the test team, JAVS capabilities, and the time frame within which testing must be completed.

10.2.1 Computer Resources

All computer resources (both hardware and support software) used by the software test object during normal execution should be identified. This information is usually contained in software documentation or can be extracted from sample execution runs. For example, for programs which execute under control of an operating system, the hardware and software requirements may be gleaned from reports produced by the system loader and information extracted from job control statements. If the program is overlayed (i.e., different parts of the program reside in main storage at different times), then the memory layout of each overlay link is also needed. This mapping of the test software onto the computer is referred to as the execution environment of the program.

Test Execution replicates normal execution in the sense that the program reads its own inputs and produces its own outputs. Additional output is captured from the instrumented program during Test Execution for later analysis by JAVS. Test Execution differs from normal execution in the following ways:

- Some or all of the program has been instrumented to capture execution behavior on a test file.
- Test case boundaries are identified at particular test points in the software.

- The instrumented code, although logically equivalent to the uninstrumented code with probes added, contains references to JAVS probe routines which are added to the load sequence.
- The probe test file is recorded.

The major effects from these software perturbations are the increased memory requirements and execution time due to code instrumentation.

10.2.2 Data Requirements

For Test Execution, the software is exercised with test data in the test environment. This data may be generated specifically for the JAVS-supported test activity, or may be taken from previous execution of the software, or both. Existing test data as well as results from tests unsupported by JAVS can be invaluable to the test process, especially if the tests reflect functional requirements of the software.

10.2.3 Test Team

Testing requires that the test team be capable of preparing data, making computer runs with the software, and analyzing output produced during execution tests. In addition, JAVS-supported testing requires that the test team use JAVS capabilities for analyzing the software, make suitable modifications for test execution, and analyze the combined test results from the software and JAVS (i.e., normal output from the software together with JAVS post-test analysis of coverage derived from software probes). It is important that the test team have more than superficial knowledge about the software test object. In particular, the team must have specification-level knowledge of functional behavior, at least limited knowledge of program structure and detailed knowledge of operational requirements.

For effective use of JAVS, the test team must understand the purpose of each of JAVS processing capabilities and select the capability applicable at each stage during the particular test activity. JAVS can be used to accelerate testing. For example, if the test team's knowledge about the software structure is deficient due to lack of detailed documentation, then, before actual testing, the JAVS documentation capability can generate the detailed information about inter-module dependencies derived directly from the software. The additional information needed about each module to attach meaning to the invocation structure includes each module's purpose, its inputs and outputs, and the interpretation associated with data processed by the module.

10.3 TEST GOALS

The overall testing goals are to improve the quality of software through testing and validation of test results. Detailed testing goals are directly related to the type of testing to be done: single-module testing or system-wide testing. The type of testing, in turn, may depend on the stage of software development and previous test history of the software test object. It is often the case that single-module testing is most appropriate during the code development phase or whenever a module has been changed or replaced during the maintenance phase. System-wide testing is applicable whenever collections of modules are tested (e.g., in software integration and maintenance phases).

10.3.1 Single-Module Testing

For single-module testing, the testing process for complete coverage has a single objective: to construct a usefully small set of test cases which, in aggregate, cause execution of each DD-path in the module at least once. Real programs may have DD-paths which cannot be exercised, no matter what input values are used. An alternate goal is to exercise each DD-path that can be exercised at least once, and for each DD-path that cannot be exercised to provide a detailed explanation why it cannot. Programs which have been tested to this level will meet the following criteria:

- Each statement in the program will have been executed at least once.
- Each decision in the program will have been brought to each of its possible outcomes at least once, although not necessarily in every possible combination.

More stringent test goals include exercising each pair of DD-paths that can be exercised in a single test, exercising all possible levels of iteration, and exercising all possible program flows (the last is, in general, not possible).

10.3.2 System-Wide Testing

For system-wide testing, the testing goal is a straightforward extension of the single-module testing goal: to construct a usefully small set of test cases which exercise as many DD-paths as possible, out of the aggregate set of DD-paths in all modules in the system. The coverage measure may be an overall percentage of DD-paths exercised, the percentage of DD-paths exercised in the least tested module, or the percentage of DD-paths exercised for the least tested subsystem of modules.

A more stringent test goal would measure coverage in relation to module location on the invocation hierarchy. For example, any modules which are referenced by more than one other module might have its coverage separately determined for each referencing module.

10.4 TESTING STRATEGY

In previous subsections the discussions have focused on defining the software test object and collecting information about its structure, operation, and the effects of using JAVS in testing. This subsection presents a general methodology for testing a software system and gives guidelines for effective JAVS usage. The sequence of major steps for testing with JAVS are:

1. Understand software system functional requirements
2. Generalize modes of system behavior
3. Define system hierarchical structure

4. Develop test plans keyed to modes of behavior and software structure

5. Execute functional tests

6. Develop structure based tests for increased coverage

10.4.1 System Behavior

Information about software (i.e., expected system response to inputs) functional requirements is usually contained in the system specification documents and software program documentation. For complex programs there may be more than a single mode of system behavior. For example, a data base management system may have a primary, high-priority mode which handles user commands interactively, and a secondary, background mode which generates periodic reports on system usage. Other modes of system behavior may include error processing or operation under degraded conditions (e.g., with an incomplete or garbled data base). If there is more than one mode of expected behavior, each should be identified and named.

10.4.2 System Structure

Having defined the system modes, an attempt should be made to relate each mode to a hierarchical structure of the program. There are several kinds of structure in a computer program: data structure is the organization of the data on which the program operates, computation structure describes the program's operations on the data, and control structure is the means of organizing the computations in the software. The control structure is the kind directly dealt with by the testing methodology. The other kinds of structure are tested only insofar as they interact with the control structure.

Large software systems are usually organized into a series of subsystems, subsystems into components, and components into modules. This state organization describes the way the individual elements of the system depend on one another without regard to the way program control flows up and down. The dynamic organization of a software system is the structure which results from considering the effects of all invocations between modules, components, and subsystems. It is usually called the invocation hierarchy of the software system. The testing methodology deals with the dynamic organization. Normally, program documentation will identify the static organization of software modules. The dynamic organization can be derived from more detailed documentation and verified with JAVS documentation capabilities showing intermodule dependencies.

For each named mode of behavior a software description should be developed which identifies the modules invoked, the specified input domain of each module, and the expected system performance. Existing software documentation may not contain sufficient information to document each behavior mode separately. It may be necessary to execute a limited number of functional tests with the software instrumented at the module-invocation level to determine which modules are invoked and under what conditions.

10.4.3 Test Plans

To minimize testing effort, test plans should be developed which are keyed to the named modes of behavior and to software structure. Each test plan should identify one or more functional tests for initial testing, as well as the software structures to be tested. It is often the case that more than one substructure of the software system will be exercised with a given test case. Such "collateral" testing can greatly reduce the overall testing effort. Each test plan should contain the following information for each functional test:

- A name identifying the test
- What function is tested
- The primary code structure tested
- Collateral code structures tested
- A description of the resources required
- The expected performance
- Criteria for evaluating the test

All these items are commonly called for in test plans for software acceptance tests. Whenever appropriate, use should be made of existing functional tests and test plans. The major distinction between testing with and without JAVS is that with JAVS there is an orderly progression of testing from the initial tests through well-defined steps to achieve the desired testing coverage, in addition to satisfying the test criteria. Quite often, additional tests to achieve increased coverage are derived from the initial functional tests.

10.4.4 Executing the Functional Tests

Before processing with JAVS, each of the initial functional tests should be used to exercise the software and to evaluate the output against the test criteria. This is important for two reasons:

- It provides a baseline set of output from the uninstrumented software for purposes of later comparison with output from the instrumented software
- It demonstrates the ability of the test team to prepare test data, execute the program, and interpret results.

This step requires that the program be complete, the source code be compiled error-free, and that test output shows acceptable execution (though not necessarily expected program behavior).

The next step is to execute each of the functional tests with instrumented software and determine initial coverage. This requires the JAVS processing to build the library (BASIC and STRUCTURAL), instrument appropriate portions of the software (INSTRUMENT), compile and execute the instrumented software with the JAVS probe routines (Test Execution), and obtain coverage reports (ANALYZER). It is very important that normal program output from Test Execution be checked against the baseline output. If discrepancies exist between the two, this is direct evidence that the addition of instrumentation has, in some way, exposed a software malfunction. Some of the reasons for this type of error are:

- The software test object is sensitive to time or space perturbations (i.e., it is not suitable for JAVS testing).
- The addition of probe routines was not properly accomplished (e.g., they were placed in the wrong overlay link).
- The test data or test environment is different from that of the baseline test.
- The computation process has caused the malfunction (e.g., using the wrong COMPOOL to compile; inconsistent code generated by the compiler).
- Computer resources are inadequate to process instrumented code.

JAVS capability for evaluating test effectiveness (ANALYZER) provides a detailed and comprehensive analysis of testing coverage. Reports on execution tracing, module and path coverage, timing, and modules and paths not exercised are generated. For the initial functional tests this information should be evaluated in some detail since it represents the point of departure for subsequent testing. Since JAVS only assists the test team in preparing the software for Test Execution, the initial coverage results may not accurately reflect actual coverage. For example, in JAVS the tester selects the placement of a "beginning of test" signal and an "end of test" signal to the probe routines and they in turn record coverage only during the user-selected interval of execution. Thus coverage reports are limited to code executed within this interval. Some reasons for unexpected coverage results are:

- The functional test does not exercise the expected modules at all.
- The selection of test point placement is improper.
- The selection of modules to instrument is not compatible with the functional test, perhaps indicating erroneous definition of system substructure.

If anomalous results occur at this point it is important to review decisions made during the previous steps in the testing process before proceeding.

10.4.5 Structure-Based Testing

Once the test team is confident of the quality achieved in obtaining initial functional test results, testing proceeds with JAVS assistance as follows:

- Selecting a testing target from information in JAVS coverage reports for previous tests
- Constructing new tests to mirror coverage using JAVS retesting assistance

- Performing Test Execution with new tests capturing software behavior data
- Analyzing test results from JAVS coverage reports

These steps are repeated until the test coverage objectives have been met.

Software testing can be performed at the single-module or system level. At the single-module level, the retesting target is a set of DD-paths which have not been exercised.

Retesting at the system level involves identification of target modules with low coverage and analysis of intermodule dependencies. Single-module testing may be viewed as part of system-wide testing. In order to construct new test cases, information about the control structure of the module and its input domain is used. The relationship of the module's input domain to the system input data must be determined in order to test the module in its normal environment (i.e., its position in the invocation structure of the software system). This may prove difficult, or not at all possible, since communication paths to the module may be blocked (e.g., by protective code or lack of knowledge of the test object). If this is the case a special test environment may be needed to thoroughly exercise the module.

10.4.6 Testing Assessment

At the conclusion of the testing an assessment should be made of the results. This summary should include the following:

- Documentation of the methods and extent of testing: strategy for testing, coverage achieved, test cases used, dynamic behavior modes tested, and identification of logically unreachable code
- Determination of the consistency between the software functional specifications: what specific functions are implemented, what unspecified functions are implemented, what unspecified restrictions are embedded
- Evaluation of existing software documentation: errors, inconsistencies, missing information, superfluous information

10.5 GENERAL STRATEGY

The best approach for systematically testing a large software system will depend on the specifics of that system's elements; it is not possible to state a universally applicable strategy. Mixtures of the top-down and bottom-up approaches may well cost the least, and may result in the greatest testing coverage.

The results of a test activity depend to a large extent on the capability and ingenuity of the test team. JAVS does offer tools not previously available to make testing more effective. Application of those tools to particular situations is the responsibility of the testers. There are however, some guidelines for selecting the most appropriate JAVS capabilities for particular situations, and some of these are described below.

Incomplete documentation. Use JAVS resources to build the library (BASIC and STRUCTURAL) and obtain documentation reports on the software (DOCUMENT). Construct missing documentation needed to start testing.

Incomplete software. Use JAVS resources to build the library (BASIC and STRUCTURAL) and obtain documentation reports on the software (DOCUMENT). To complete test environment, first identify top-level modules and external library dependencies. Next, construct a driver for each top-level module: Use JAVS module invocation definition and cross reference for calling protocol and input domain. Provide stubs (i.e., dummy modules) for externals which are referenced but are not present on system or auxiliary libraries: Use JAVS module invocation references for calling protocol. Documentation supplied with the software should be consulted for module interface specifications.

Single-module testing. Use unexercised DD-paths report (ANALYZER, NOTHIT)* to identify potential test target paths. To get the control nesting level of unexercised DD-paths, use the module listing (PRINT,MODULE)* or DD-path definitions report (PRINT,DDPATHS).* Use control flow picture (ASSIST,PICTURE)* for an overview of module structure. Select testing target from reaching set for target path (ASSIST,REACHING SET)* and determine module inputs which cause target path to execute. Generate test data (see below).

System-wide testing. Use module coverage summary and DD-path coverage summary to identify potential test target modules. For any module never invoked use inter-module dependencies (DOCUMENT) to determine what modules cause it to be invoked directly and indirectly through other modules. Identify which higher level modules were executed. Using cross reference reports together with inter-module dependence reports to identify what modules affect invocation, modify test data to cause invocation of target module. Several cycles of top-down testing may be required if the unexercised inter-module control structure is at all complex. Apply single-module testing techniques to increase intra-module coverage.

Unknown behavior. Plant document probes (PROBD) to capture imbedded descriptive information in test execution trace. Examine test trace report to identify behavior with probe location. Use results to select appropriate test points.

Unexpected behavior. Use JAVS assertion statements to isolate causes. At the beginning of the module, insert assertion statements for expected condition of module inputs. At the end of module, where control is returned, insert assertion statements for expected

*See Sec. 5, JAVS Reference Manual, under the command headings.

conditions of module outputs. At intermediate locations in the module, insert assertion statements for expected conditions of module behavior. Examine Test Execution output for report of unexpected behavior.

There are several side benefits to be realized from testing. For example, the software can be optimized by removal of unreachable code or code which implements extraneous functions. JAVS documentation reports are useful here in determining the extent of changes to the software (e.g., modules and data structures affected) and the amount of retesting necessary after software modifications have been made.

APPENDIX A
JAVS COMMAND SUMMARY

JAVS COMMANDS (DEFAULTS UNDERLINED)	STEP
ALTER LIBRARY = <libname>.	(Universal)
ANALYZER.	ANALYZER
ANALYZER,ALL.	ANALYZER
ANALYZER,ALL MODULES.	ANALYZER
ANALYZER,CASES = <number>.	ANALYZER
ANALYZER,DDPATHS.	ANALYZER
ANALYZER,DDPTRACE.	ANALYZER
ANALYZER,FACTOR = <percent-increase>.	ANALYZER
ANALYZER,HIT.	ANALYZER
ANALYZER,MODLST.	ANALYZER
ANALYZER,MODTRACE.	ANALYZER
ANALYZER,MODULE = <name-1>,<name-2>,...,<name-n>.	ANALYZER
ANALYZER,NOTHIT.	ANALYZER
ANALYZER,SUMMARY.	ANALYZER
ANALYZER,TIME.	ANALYZER
ASSIST,CROSSREF,JAVSTEXT = <text-name-1>,<text-name-2>,..., <text-name-n>.	ASSIST
ASSIST,CROSSREF,LIBRARY.	ASSIST
ASSIST,PICTURE.	ASSIST
ASSIST,PICTURE{,CONTROL}{,NOSWITCH}.	ASSIST
ASSIST,REACHING SET,<number-to>{,<number-from>} {,PICTURE{,ITERATIVE}}.	ASSIST
ASSIST,STATEMENTS.	ASSIST
BASIC.	BASIC
BASIC,CARD IMAGES = ON/OFF.	BASIC
BASIC,COMMENTS = ON/OFF.	BASIC
BASIC,DEFINES = ON/OFF.	BASIC
BASIC,ERRORS = ON/OFF/LIMIT/TRACE.	BASIC
BASIC,SYMBOLS = ON/OFF/PARTIAL.	BASIC
BASIC,TEXT = COMPUTE/BOTH/PRESET/JAVSTEXT.	BASIC
*BUILD LIBRARY {= <library name>}.	BASIC, STRUCTURAL
CREATE LIBRARY = <libname>.	(Universal)

*Can be used only with the overlay version.

JAVS COMMANDS (DEFAULTS UNDERLINED)	STEP
DEPENDENCE,BANDS.	DEPENDENCE
DEPENDENCE,BANDS = <number>.	DEPENDENCE
DEPENDENCE,GROUP,AUXLIB.	DEPENDENCE
DEPENDENCE,GROUP,LIBRARY.	DEPENDENCE
DEPENDENCE,GROUP,MODULES = <name-1>,<name-2>,...,<name-n>.	DEPENDENCE
DEPENDENCE,PRINT,INVOKES.	DEPENDENCE
DEPENDENCE,SUMMARY.	DEPENDENCE
DEPENDENCE,TREE.	DEPENDENCE
DESCRIBE = ON/OFF.	(Universal)
*DOCUMENT{,JAVSTEXT=<text-name>{,MODULE=<name-1>,...}}.	ASSIST, DEPENDENCE, (Universal)
END.	(Universal)
END FOR.	(Universal)
FOR JAVSTEXT.	(Universal)
FOR LIBRARY.	(Universal)
FOR MODULE = <name-1>,<name-2>,...,<name-n>.	(Universal)
INSTRUMENT.	INSTRUMENT
INSTRUMENT,MODE = INVOCATION/DDPATHS/DIRECTIVES/FULL.	INSTRUMENT
INSTRUMENT,PROBE,DDPATH = <probe-name>.	INSTRUMENT
INSTRUMENT,PROBE,MODULE = <invocation-name>.	INSTRUMENT
INSTRUMENT,PROBE,TEST = <test-name>.	INSTRUMENT
INSTRUMENT,STARTTEST = <modname>,<textname>,<stmt. no.> {,<TESNAM>}{,<TFLAG>}.	INSTRUMENT
INSTRUMENT,STOPTEST = <modname>,<textname>, <stmt. no.>.	INSTRUMENT
JAVSTEXT = <text-name>.	(Universal)
MERGE.	(Universal)
MODULE = <name>.	(Universal)
OLD LIBRARY = <libname>.	(Universal)

*Can be used only with the overlay version.

JAVS COMMANDS (DEFAULTS UNDERLINED)	STEP
PRINT,DDP.	(Universal)
PRINT,DDPATHS.	(Universal)
PRINT,DMT.	(Universal)
PRINT,JAVSTEXT = <text-name-1>, INSTRUMENTED = ALL.	(Universal)
PRINT,JAVSTEXT = <text-name>,INSTRUMENTED = <name-1>, <name-2>,...,<name-n>.	(Universal)
PRINT,JAVSTEXT = <text-name>.	(Universal)
PRINT,MODULE.	(Universal)
PRINT,SB.	(Universal)
PRINT,SDB.	(Universal)
PRINT,SLT.	(Universal)
PRINT,STB.	(Universal)
*PROBE{,JAVSTEXT = <text-name>{,MODULE = <name-1>,...}}.	INSTRUMENT, (Universal)
*PROBI,STARTTEST = <modname>,<textname>,<stmt. no.> {,TESNAM}{,TFLAG}.	INSTRUMENT, (Universal)
*PROBI,STOPTEST = <modname>,<textname>,<stmt. no.>.	INSTRUMENT, (Universal)
PUNCH,JAVSTEXT = <text-name>.	(Universal)
PUNCH,JAVSTEXT = <text-name>,INSTRUMENTED = ALL.	(Universal)
PUNCH,JAVSTEXT = <text-name>,INSTRUMENTED = <name-1>, <name-2>,...,<name-n>.	(Universal)
PUNCH,MODULE.	(Universal)
START.	(Startup)
STRUCTURAL.	STRUCTURAL
STRUCTURAL,PRINT = SUMMARY/DEBUG.	STRUCTURAL
*TEST{,MODULE = <name-1>,<name-2>,...<name-n>}.	ANALYZER, (Universal)

---
*Can be used only with the overlay version.

APPENDIX B

JAVS MACRO COMMANDS

B.1 INTRODUCTION

To facilitate the use of JAVS, four new commands were added to the existing set of processing commands. These "macro" commands combine the most commonly used commands from the standard set and must be used with the linked version of JAVS. The macro commands may be used one at a time, all together, or in combination with the standard set of commands. The combination of commands requires understanding the expansion of commands which each macro generates; thus, the user is urged to review Sec. B.2 carefully. The four macro commands are:

1. BUILD LIBRARY [=<name>].
2. PROBE [,JAVSTEXT = <text-name>[,MODULE = <name-1>, <name-2>, ...<name-n>]].
3. TEST[,MODULE = <name-1>, <name-2>, ..., <name-n>].
4. DOCUMENT [,JAVSTEXT = <text-name>[,MODULE = <name-1>, <name-2>, ... <name-n>]].

Brackets [] indicate optional information. Each option generates a different set of standard commands.

B.2 EXPANSION OF MACRO COMMANDS

Unless the user supplies the library identification and start commands, the first occurrence of a macro command in the command set generates the standard commands:

```
CREATE LIBRARY = TEST.
START.
```

or

```
OLD LIBRARY = TEST.
START.
```

All macros except BUILD LIBRARY generate the OLD LIBRARY command.

B.2.1 Syntax and Structural Analysis

BUILD LIBRARY. generates commands:

```
CREATE LIBRARY = TEST.
START.
BASIC, COMMENTS = OFF.
BASIC.
FOR LIBRARY.
STRUCTURAL.
END FOR.
```

BUILD LIBRARY = <name>. generates commands:

```
CREATE LIBRARY = <name>.
START.
BASIC,COMMENTS = OFF.
BASIC.
FOR LIBRARY.
STRUCTURAL.
END FOR.
```

B.2.2 Documentation Reports

DOCUMENT. generates commands:

```
ASSIST, CROSSREF, LIBRARY.
DEPENDENCE, GROUP, LIBRARY.
DEPENDENCE, GROUP, AUXLIB.
DEPENDENCE, SUMMARY.
FOR LIBRARY.
PRINT,MODULE.
DEPENDENCE, BANDS=5
DEPENDENCE, PRINT, INVOKES.
END FOR.
```

DOCUMENT, JAVSTEXT = <text name>. generates commands

```
ASSIST, CROSSREF, LIBRARY.
DEPENDENCE, GROUP, LIBRARY.
DEPENDENCE, GROUP, AUXLIB.
DEPENDENCE, SUMMARY.
JAVSTEXT = <text name>.
FOR JAVSTEXT.
PRINT, MODULE.
DEPENDENCE, BANDS = 5.
DEPENDENCE,PRINT,INVOKES.
END FOR.
```

DOCUMENT, JAVSTEXT = <text-name>, MODULE = <name-1>, ..., <name-n>. generates commands:

```
ASSIST, CROSSREF, LIBRARY.
DEPENDENCE, GROUP, LIBRARY.
DEPENDENCE, GROUP, AUXLIB.
DEPENDENCE, SUMMARY.
JAVSTEXT = <text name>.
FOR MODULE = <name-1>, <name-2>, ..., <name-n>.
PRINT, MODULE.
DEPENDENCE, BANDS = 5.
DEPENDENCE, PRINT, INVOKES.
END FOR.
```

B.2.3 Instrumentation

PROBE. generates commands:

```
JAVSTEXT = *NOJAVS*.
FOR JAVSTEXT.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = *NOJAVS*, INSTRUMENTED = ALL.
```

PROBE, JAVSTEXT = <text name>. generates commands:

```
JAVSTEXT = <text name>.
FOR JAVSTEXT.
INSTRUMENT.
END FOR.
PUNCH, JAVSTEXT = <text name>, INSTRUMENTED = ALL.
```

PROBE, JAVSTEXT = <text name>, MODULE = <name-1>, <name-2>, ..., <name-n>. generates commands:

```
JAVSTEXT = <text name>.
FOR MODULE = <name-1>, ..., <name-n>.
INSTRUMENT.
END FOR.
PUNCH, JAVSTEXT = <text name>, INSTRUMENTED = <name-1>,
..., <name-n>.
```

B.2.4 Test Boundary Insertion (quasi-macro commands)

In order to identify test cases and control the recording of data on the AUDIT file, the user must supply invocations to the data collection routine, PROBI. The invocations can be manually inserted prior to Test Execution, or they can be automatically inserted during instrumentation.

The PROBI commands cause JAVS to insert an invocation to PROBI for identifying a new test case or terminating the test. The commands are of the form:

```
PROBI,STARTTEST =  <m-name>, <t-name>, <no.>{,<TESNAM>,<TFLAG>}.
PROBI,STOPTEST = <m-name>, <t-name>, <no.>.
```

where

```
m-name = module name
t-name = Javstext name
   no. = statement number
TESNAM = test case identifier   DEFAULT = 8H(CASE   )
 TFLAG = tracing level          DEFAULT = 2
```

The command options must be given in the order shown above. The user must specify the module and JAVSTEXT names in which the PROBI invocation is to be inserted. The user must also specify the statement number (using the statement number presented in the JAVS module listing) before which the invocation is to be inserted. The user may specify the statement number to be 0. This results in placing the PROBI,STARTTEST call immediately prior to the first software probe (at the first executable statement) and the PROBI,STOPTEST call immediately preceeding each exit from the module.

TESNAM and TFLAG are the two input parameters to PROBI. If these parameters are not specified in this command, the default values will be used, causing tracing of module invocations and returns. TESNAM may be up to eight characters; TFLAG may be 1, 2, or 3. A maximum of ten PROBI,STARTTEST commands can be used for a single PROBE command. The module and JAVSTEXT names may be different in each command.

The PROBI commands, if used, must precede the PROBE macro command(s). Any module specified in a PROBI command must be specified in the PROBE macro command.

B.2.5 Post-Test Analysis

TEST. generates commands:

```
ANALYZER, ALL MODULES.
ANALYZER, SUMMARY.
ANALYZER, NOTHIT.
ANALYZER, MODTRACE.
ANALYZER.
```

TEST, MODULE = <name-1>, <name-2>, ..., <name-n>. generates commands:

```
ANALYZER, MODULE = <name-1>, ..., <name-n>.
ANALYZER, SUMMARY.
ANALYZER, NOTHIT.
ANALYZER, MODTRACE.
ANALYZER.
```

Note:

After the last command is read by the macro command processor, an "END." standard command is generated.

APPENDIX C

JAVS FILES

C.1 INTRODUCTION

The files used in JAVS processing are listed in Table C.1 together with important characteristics about each file. On systems which allocate files by number (e.g., GCOS) the file number is used; on those which allocate the file by name (e.g., GOLETA), the file name is used. The data structure column indicates the contents of the file. The mode indicates how JAVS references the file. The storage form and record format describe how the data is recorded. The recommended allocation suggests an appropriate type of system file, keeping in mind that random files must be on direct access devices and sequential files may be on either direct access devices or serial devices. The usage indicates how each file is utilized for different types of JAVS processing.

The JAVS Reference Manual[1] contains a detailed description of file usage for each processing step.

C.2 RANDOM FILES

LIBOLD and LIBNEW are used for the JAVS data base library. LIBWSP is always used for working space. On all of these random files, the JAVS Storage Manager allocates space for each JAVS table in contiguous groups of 500 words called "fragments." Each file is treated internally as a word-addressable file, although it may be recorded in another form (e.g., as fixed-length records). The wrapup summary at the end of each JAVS execution contains the current size for each of these files.

C.3 SEQUENTIAL FILES

COMMAN and COMAUX are used for JAVS commands. COMMAN has a card image record for each command line; COMAUX (always shorter than COMMAN) is used to store the commands within an iteration sequence. COMMAN must always be allocated for any processing step. COMAUX must be allocated whenever any FOR command is used.

LOUT contains all JAVS reports destined for the line printer. The number of records on LOUT depends on the number and types of reports produced. The example reports in the JAVS Reference Manual are useful in estimating the size of LOUT. LOUT is always needed in any processing step.

LPUNCH contains instrumented (or non-instrumented) source (in card image form) destined for the JOVIAL compiler. The card image source can be written at any time as long as it was saved on the database library. Usually, LPUNCH is written during the instrumentation activity.

AUDIT contains the Test Execution probe data. It is possible to record three types of records on AUDIT. The types of records actually recorded can be controlled during INSTRUMENT processing by the MODE option and during Test Execution by an argument value to PROBI. Clearly, for a fully instrumented program with complete tracing and a large number of DD-path executions, the number of records on AUDIT can become very large (to say nothing about the added processing time). AUDIT is used in Test Execution and ANALYZER. In general there is no way to estimate the size of the AUDIT file, since it

TABLE C.1

FILES USED IN JAVS PROCESSING

FILE NUMBER	FILE NAME	DATA STRUCTURE	MODE (1)	STORAGE FORM (2)	RECORD FORMAT	RECOMMENDED ALLOCATION	PROCESSOR USAGE (3) UNIVERSAL COMMANDS	BASIC	STRUCTURAL	INSTRUMENT	ASSIST	DEPENDENCE	TEST EXECUTION	ANALYZER
1 (4)	LIBOLD	library	B	R	system standard (6)	permanent file (8)	R	R	R	R	R	R		R
2 (5)	LIBNEW	library	B	R	system standard (6)	permanent file (8)	R/W	R/W	R/W	R/W	R/W	R/W		R/W
3	LIBWSP	workspace	B	R	system standard (6)	scratch file (9)	R/W[4]	R/W[4]	R/W[4]	R/W	R/W	R/W		R/W
4 (10)	COMAUX	iteration commands	H	S	card image	scratch file (9)		R/W	R/W	R/W	R/W	R/W		R/W
5	COMMAN	user commands	H	S	card image	system card reader	R	R	R	R	R	R		R
6	LOUT	reports	H	S	128 characters/ line maximum	system printer	W	W	W	W	W	W	W(7)	W
7 (11)	LPUNCH	instrumented source	H	S	card image	system punch/ permanent file (8)	W							
8	AUDIT	probe test data	B	S	8 machine words	permenent file (8)/ magnetic tape							W	R
9	READER	JOVIAL source	H	S	card image	card file/ permanent file (8)		R						
10 (12)	COMMAC	user commands	H	S	card image	scratch file (9)	R/W	R/W	R/W	R/W	R/W	R/W		R/W

Notes:
(1) B = binary; H = BCD
(2) R = random; S = sequential
(3) R = read only; R/W = read and/or write; W = write only
(4) required with OLD LIBRARY command
(5) required with CREATE LIBRARY or ALTER LIBRARY command; required for ASSIST and ANALYZER
(6) installation dependent
(7) output on standard print file
(8) permanent file must be previously created by a FILSYS activity (see GCOS File System reference manual or Control Cards Reference Manual, under $PRMFL)
(9) mass storage for scratch file (see GCOS reference manual or Control Cards Reference Manual, under $FILE)
(10) required with any FOR command
(11) required with any PUNCH command
(12) required with overlay version of JAVS; not required with standard (non-overlay) version

depends on the execution behavior of the program being analyzed.

READER contains the JOVIAL source (in card image form) to be analyzed by JAVS. READER is used by the Syntax Analyzer (BASIC).

COMMAC is an intermediate command file used by the macro command processor. COMMAC is required in all processing steps whenever the JAVS overlay version is used.

APPENDIX D

SAMPLE JOB STREAMS FOR RADC

D.1 PERFORM SYNTAX AND STRUCTURAL ANALYSES

```
$      IDENT     <userid>,<user name>,<acc't. no.>
$      USERID    <userid>$<password>
$      SELECT    BFCBGRC1/STEP1-6
$      PRMFL     02, R/W,R,<userid>/<library file name>
$      PRMFL     09, R,S,<userid>/<source file name>
BUILD LIBRARY.
$      ENDJOB
```

Notes:

(1) SELECT stream STEP1-6 is required by the syntax analyzer. All other JAVS activities can use the smaller STEP2-6.

(2) The LIMITS control card imbedded in the SELECT stream specifies .99 CP hour and 40,000 lines of output. The user can modify these limits by placing the following control card before the JAVS command:

```
$      LIMITS     <time>,82K,-5K,<lines>
```

(3) To process JOVIAL comments (they cannot be imbedded within a JOVIAL statement), replace BUILD LIBRARY with:

```
CREATE LIBRARY = <library name>.*
START.
BASIC.
FOR LIBRARY.
STRUCTURAL.
END FOR.
```

(4) To obtain the JAVS enhanced listing for each module during this activity, follow BUILD LIBRARY with:

```
FOR LIBRARY.
PRINT,MODULE.
END FOR.
```

(5) To obtain JAVS documentation reports within the same activity, follow BUILD LIBRARY with:

```
DOCUMENT.
```

---

*The default library name is TEST. The user may provide any library name (up to eight characters) but in doing so must provide the library identification and start commands in subsequent JAVS processing jobs to reflect the non-default library name. See page B-2 for more information.

D.2 OBTAIN JAVS DOCUMENTATION REPORTS

```
$      IDENT      <userid>,<user name>,<acc't. no.>
$      USERID     <userid>$<password>
$      SELECT     BFCBGRC1/STEP2-6
$      PRMFL      01, R,R,<userid>/<library file name>
DOCUMENT.
$      ENDJOB
```

Notes:

(1) The LIMITS control card imbedded in the SELECT stream specifies .60 CP hour and 20,000 lines of output. The user can modify these limits by placing the following control card before the JAVS command:

```
$      LIMITS      <time>,55K,-5K,<lines>
```

(2) To obtain the JAVS control flow picture for each module, in addition to the other documentation reports, follow the DOCUMENT command with:

```
FOR LIBRARY.
ASSIST,PICTURE.
END FOR.
```

or precede the DOCUMENT command with:

```
OLD LIBRARY = TEST.*
START.
FOR LIBRARY.
ASSIST,PICTURE.
END FOR.
```

*The default library name is TEST. The user may provide any library name (up to eight characters) but in doing so must provide the library identification and start commands in subsequent JAVS processing jobs to reflect the non-default library name. See page B-2 for more information.

D.3 INSTRUMENT A START-TERM SEQUENCE (JAVSTEXT)*

```
$     IDENT       <userid>,<user name>,<acc't. no.>
$     USERID      <userid>$<password>
$     SELECT      BFCBGRC1/STEP2-6
$     PRMFL       01, R,R,<userid>/<library file name>
$     PRMFL       07, W,S,<userid>/<instrumented source file name>
OLD LIBRARY = TEST.**
START.
PROBI,STARTTEST = <options>.
PROBI,STOPTEST = <options>.
PROBE,JAVSTEXT = <text name>.
$     ENDJOB
```

Notes:

(1) See D.2 note (1).

(2) Library identification and start commands are required for PROBI commands.

(3) To manually insert the PROBI test case boundary calls, remove the first four commands.

(4) To obtain a listing of the instrumented modules, follow the PROBE command with:

```
PRINT,JAVSTEXT = <text name>,INSTRUMENTED = ALL.
```

---

* This job stream does not save the probed code on the database library.

** The default library name is TEST. The user may provide any library name (up to eight characters) but in doing so must provide the library identification and start commands in subsequent JAVS processing jobs to reflect the non-default library name. See page B-2 for more information.

D.4 INSTRUMENT A START-TERM SEQUENCE (JAVSTEXT)*

```
$      IDENT       <userid>,<user name>,<acc't. no.>
$      USERID      <userid>$<password>
$      PROGRAM     RLHS
$      PRMFL       H*,R,R,BFCBGRC1/MJAVS2-6
$      PRMFL       02, R/W,R,<userid>/<library file name>
$      PRMFL       07, W,S,<userid>/<instrumented source file name>
$      FILE        10,C1R,1L
$      FILE        03,X3R,<library file size>
$      FILE        04,X4R,2L
$      LIMITS      <time>,55K,-5K,<lines>
ALTER LIBRARY = TEST.**
START.
PROBI,STARTTEST = <options>.
PROBI,STOPTEST =  <options>.
PROBE,JAVSTEXT =  <text name>.
$      ENDJOB
```

See D.3 notes.

*This job stream saves the probed code on the database library which can substantially increase the library's file size.

**The default library name is TEST. The user may provide any library name (up to eight characters) but in doing so must provide the library identification and start commands in subsequent JAVS processing jobs to reflect the non-default library name. See page B-2 for more information.

D.5 COMPILE INSTRUMENTED SOURCE TEXT

```
$     IDENT      <userid>,<user name>,<acc't. no.>
$     USERID     <userid>$<password>
$     LOWLOAD
$     OPTION     FORTRAN
$     JOVIAL     NOPT,NDECK,NAME/PRPOOL/,POOLOU/JP/
$     FILE       JP,X1S,2L
$     SELECT     BFCBCS01/COMPILEB
$     LIMITS     1,49K
$     SELECT     BFCBGRC2/PRPOOL
$     JOVIAL     POOLIN/JP{,user COMPOOLS}/,NOPT,
$     ETC        NAME/<name>/
$     FILE       JP,X1R,2L
$     SELECT     BFCBCS01/COMPILEB
$     LIMITS     <time>,<size>,,<lines>
$     PRMFL      S*,R,S,<userid>/<instrumented source file>
$     PRMFL      C*,W,S,<userid>/<instrumented object file>
.
.     user COMPOOL perm files
.

$     ENDJOB
```

Notes:

(1) Currently the backup JOCIT compiler (COMPILEB) is being used.

(2) The CP time in the second LIMITS control card should be approximately 1.5 times the CP time required to compile the uninstrumented text.

(3) The core size in the second LIMITS control card should be approximately 1.25 times the size required to compile the uninstrumented text.

(4) The instrumented source code may contain a $ in column one (1). In this event, replace the PRMFL S* control card with:

```
$    DATA       S*,,COPY,ENDFC
$    SELECTD    <userid>/<instrumented source file>
$    ENDCOPY    S*
```

D.6 TEST EXECUTION AND POST-TEST ANALYSIS

```
$      IDENT      <userid>,<user name>,<acc't. no.>
$      USERID     <userid>$<password>
$      LOWLOAD
$      OPTION     FORTRAN
$      SELECT     BFCBGRC1/JPROBESX
$      SELECT     <userid>/<instrumented object file>
$      SELECT     BFCBCS01/EXECUTEB
$      LIMITS     <time>,<size>,,<lines>
$      PRMFL      08, W,S,<userid>/<AUDIT file>
.
.      user files and data
.
$      SELECT     BFCBGRC1/STEP2-6
$      LIMITS     <time>, 55K,-5K,<lines>
$      PRMFL      01, R,R,<userid>/<library file>
$      PRMFL      08, R,S,<userid>/<AUDIT file>
OLD LIBRARY = TEST.*
START.
ANALYZER,MODLST.
ANALYZER,DDPATHS.
TEST.
$      ENDJOB
```

Notes:

(1) To obtain a printed listing of the input data, if the data were on a perm file, insert the following control cards before selecting STEP2-6:

```
$   CONVER
$   INPUT     NMEDIA
$   PRMFL     IN,R,S,<userid>/<data file>
$   PRINT     OT
```

(2) The EXECUTEB select stream supplies JOVIAL system routines used by the backup JOCIT compiler. This can be changed to EXECUTE if all software modules were compiled using a newer version of JOCIT.

(3) Additional instrumented files can be loaded.

(4) In an overlay environment, JPROBESX must be loaded in the main link.

(5) The AUDIT file can be a scratch disk file or magnetic tape, instead of a perm file.

---

*The default library name is TEST. The user may provide any library name (up to eight characters) but in doing so must provide the library identification and start commands in subsequent JAVS processing jobs to reflect the non-default library name. See page B-2 for more information.

D.7 RETESTING ASSISTANCE (REACHING SETS)

```
$      IDENT      <userid>,<user name>,<acc't. no.>
$      USERID     <userid>$<password>
$      SELECT     BFCBGRC1/STEP2-6
$      PRMFL      01, R,R,<userid>/<library file name>
OLD LIBRARY = TEST.*
START.
ASSIST,REACHING SET,<target>,{options}.
.
.
.
$      ENDJOB
```

---

* The default library name is TEST. The user may provide any library name (up to eight characters) but in doing so must provide the library identification and start commands in subsequent JAVS processing jobs to reflect the non-default library name. See page B-2 for more information.

D.8 SELECT STREAMS

In the event that the user wishes to modify the control cards nested in the three JAVS SELECT control cards, the expansions are as follows:

```
$     SELECT BFCBGRC1/STEP1-6                contains

$     PROGRAM RLHS
$     PRMFL   H*,R,R,BFCBGRC1/MJAVS1-6
$     LIMITS  99, 82K, -5K, 40000
$     FILE    03, X3R, 10R
$     FILE    10, C1R, 1L
$     FILE    04, X4R, 2L
```

```
$     SELECT BFCBGRC1/STEP2-6                contains

$     PROGRAM RLHS
$     PRMFL   H*,R,R,BFCBGRC1/MJAVS2-6
$     FILE    10, C1R, 1L
$     FILE    02, X2R, 20R
$     LIMITS  60, 55K, -5K, 20000
$     FILE    03, X3R, 10R
$     FILE    04, X4R, 2L
```

```
$     SELECT BFCBGRC1/JPROBESX               contains

$     SELECT  BFCBGRC2/BPRCMPL
$     SELECT  BFCBGRC2/BPROBE
$     SELECT  BFCBGRC2/BPROBI-X
$     SELECT  BFCBGRC2/BPROBM-X
$     SELECT  BFCBGRC2/BPROBD-X
```

APPENDIX E

TIME AND SIZE ESTIMATIONS

TABLE E.1

FILE SIZE ESTIMATION

File #	File	Estimation*
01, 02	Library (LIBOLD, LIBNEW)	15-20 llinks/module 300 llinks/1000 source statements 4-5 times source file size (llinks)
03	LIBWSP	10 llinks
04	COMAUX	2 llinks
07	LPUNCH (instrumented source)	8 llinks/module 100 llinks/1000 source statements 2 times uninstrumented source file size (llinks)
	Instrumented object file	.3 times LPUNCH (llinks) 2 times uninstrumented object file size
08	AUDIT	Minimum size (no execution tracing) is: (number of probed DD-paths x number of test cases x .2 llinks) Maximum size is dependent upon execution behavior and level of tracing
09	READER	.04 llinks/source card
10	COMMAC	1 llink

*These estimations are derived from testing the SAC Force Management Information System.

TABLE E.2

CP Time Estimation

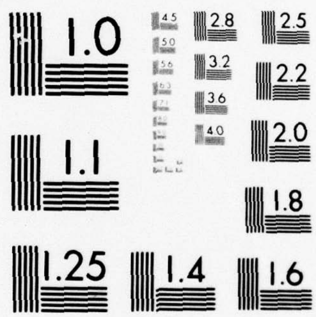
Task/Command	CP Hour/Module	CP Hour/ 1000 Statements
BUILD LIBRARY	.006	.1
DOCUMENT	.006	.1
PROBE	.005	.07
Compile instrumented code	.001	.02
Test Execution	1.5 times execution time for uninstrumented program	
TEST*	3-6 times Test Execution time .01 CP hour/module	

*Very rough estimates, since TEST CP time depends heavily on the size of the AUDIT file.

AD-A040 103 GENERAL RESEARCH CORP SANTA BARBARA CALIF F/G 9/2
JAVS TECHNICAL REPORT. USER'S GUIDE.(U)
APR 77 C GANNON, N B BROOKS, R J URBAN F30602-76-C-0233
UNCLASSIFIED RADC-TR-77-126-VOL-1 NL

2 OF 2
AD
A040103

END
DATE
FILMED
6-77



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

APPENDIX F

JAVS INSTALLATION REQUIREMENTS

TABLE F.1

JAVS INSTALLATION AT RADC

DATE	October 1976
VERSION	2.0 non-overlay, 2.0 overlay
COMPUTER	HIS 6180
OPERATING SYSTEM	GCOS version G update 3
COMPILER	JOCIT JOVIAL/J3 version 042275
NON-STANDARD FEATURES	JOVIAL MONITOR, FORTRAN random I/O
CONFIGURATION	batch, unlinked; batch, linked

PROCESSING CORE REQUIREMENTS:

Program File	Type	Load Size	Process
MJAVS1-6	H*	81K	Complete JAVS overlay
MJAVS2-6	H*	54K	JAVS overlay w/o BASIC
EXECS1	H*	84K	BASIC
EXECS2	H*	84K	STRUCTURAL
EXECS3	H*	57K	INSTRUMENT
EXECS4	H*	54K	ASSIST
EXECS5	H*	53K	DEPENDENCE
EXECS6	H*	54K	ANALYZER
JPROBESX	select	4K	Test Execution

FILES:

Number	Name	Allocation	Usage	Description (1)
1	LIBOLD	save	R/W	300 W/records, R, U, F
2	LIBNEW	save	R/W	300 W/record, R, U, F
3	LIBWSP	scratch	R/W	300 W/record, R, U, F
4	COMAUX	scratch	R/W	BCD, card image
(2) 5	COMMAN	card reader	R	system input, BCD
(3) 5	COMMAC	card reader	R	system input, BCD
6	LOUT	printer	W	system output
7	LPUNCH	compile	W	BCD, card image
8	AUDIT	save	R/W	Binary, 8 W/record
9	READER	source	R	BCD, card image
(3)10	COMMAN	scratch	R/W	BCD, card image

Notes:
(1) W = words, R = random, U = unpartioned, F = fixed length
(2) JAVS 2.0 non-overlay version
(3) JAVS 2.0 overlay version

TABLE F.2

JAVS INSTALLATION AT SAC HEADQUARTERS

DATE	January 1977
VERSION	2.0 overlay
COMPUTER	HIS 6180
OPERATING SYSTEM	WWMCCS
COMPILER	JOCIT JOVIAL/J3 version 042275
NON-STANDARD FEATURES	JOVIAL MONITOR, FORTRAN random I/O
CONFIGURATION	batch, linked

PROCESSING CORE REQUIREMENTS:

Program File	Type	Load Size	Process
MJAVS1-6	H*	81K	Complete JAVS overlay
MJAVS2-6	H*	54K	JAVS overlay w/o BASIC
JPROBESX	select	4K	Test Execution

FILES:

Number	Name	Allocation	Usage	Description (1)
1	LIBOLD	save	R/W	300 W/records, R, U, F
2	LIBNEW	save	R/W	300 W/record, R, U, F
3	LIBWSP	scratch	R/W	300 W/record, R, U, F
4	COMAUX	scratch	R/W	BCD, card image
5	COMMAC	card reader	R	system input, BCD
6	LOUT	printer	W	system output
7	LPUNCH	compile	W	BCD, card image
8	AUDIT	save	R/W	Binary, 8 W/record
9	READER	source	R	BCD, card image
10	COMMAN	scratch	R/W	BCD, card image

Note:

(1) W = words, R = random, U = unpartioned, F = fixed length.

APPENDIX G

JAVS UTILIZATION CHECKLIST

1. Prepare source code:

   a. Insert JAVSTEXT directive as the first statement of each START-TERM sequence.

      If the START-TERM is a program, CLOSE, or PROC use:

      ".JAVSTEXT <name> COMPUTE (<COMPOOL name>)"

      The parenthetical name informs JAVS that one or more COMPOOLs are referenced, although the COMPOOL name is not checked for validity.

      If the START-TERM is a COMPOOL use:

      ".JAVSTEXT <name> PRESET"

      See page 3-5 in the User's Guide and page 1-7 in the Reference Manual for examples.

      Only one COMPOOL can be put on the JAVS data base Library for each execution of the BASIC processing step (syntax analysis). COMPOOLs must not be processed by the structural analyzer (STRUCTURAL command keyword), thus the BUILD LIBRARY macro command cannot be used when processing COMPOOLS and executable source code together.

   b. If any START-TERM sequence is an external CLOSE (i.e., the first module following the START is a CLOSE), remove the CLOSE <name> statement before inputting the source code to JAVS. Insert the CLOSE statement in the instrumented source code before compilation.

   c. If JAVS computation directives (ASSERT, EXPECT, TRACE, OFFTRACE) are to be used, insert them into the source code following normal JOCIT programming rules for placement and expression syntax. See Sec. 1.5 in the Reference Manual for description of these directives.

2. Create files

   a. Complete JAVS processing of the source code will require creation of the following files:

File code	Name	Type	Contents
01,02	LIBNEW LIBOLD	Random	JAVS data base library
07	LPUNCH	Sequential	Instrumented source
08	AUDIT	Sequential	Execution Trace

      See page E-2 of the User's Guide for size estimations of these files.

   b. Additional files which the user may wish to use are the sequential C* file containing the instrumented object code (see page D-6, User's Guide) and a random access H* file containing the user's program with instrumented code.

3. Perform syntax and structural analyses

   a. Execute the job stream on page D-2, User's Guide (or one which employs BASIC and STRUCTURAL keywords; see Sec. 5 of the Reference Manual under these keyword headings).

   b. Check JAVS output for any errors. The complete list of errors is in Appendix B of the Reference Manual.

   c. If necessary, modify source and reprocess this step.

4. Obtain JAVS documentation reports

   - Execute the job stream on page D-3, User's Guide or

   - Use any of the PRINT, ASSIST and DEPENDENCE commands to produce the desired documentation reports. See Sec. 5 of the Reference Manual under the appropriate keyword headings for sample commands and output.

5. Instrument the source code

   a. For each START-TERM (JAVSTEXT) execute the job stream on page D-4, User's Guide.

   b. The PROBI commands direct JAVS to insert calls to the PROBI data collection routine to initiate and terminate test cases at specified statements in the source code (statement numbers appear in the JAVS module listings). See Sects. 5.3 and 6, User's Guide and page 2-15, Reference Manual for PROBI description. The test case initiation and termination PROBI calls can be inserted manually (e.g., under the GCOS EDIT system) in the <u>instrumented or uninstrumented</u> source code, or they can be inserted at the direction of the PROBI command. See page 2-17, Reference Manual for a sample listing of probed code.

6. Compile the instrumented source text

   a. Use the job stream on page D-6, User's Guide, supplying any COMPOOLs (in addition to the JAVS PRPOOL) required for compilation.

   b. JAVS instrumentation modifies certain control statements (e.g., an IF becomes an IFEITH). This can cause a statement to be continued on the next line. If the continuation line contains a $ in column 1, GCOS and WWMCCS will treat it as a control card. If this occurs, use the control cards given in note (4), page D-6.

7. Load, execute and analyze program coverage

   a. Use the job stream on page D-7, User's Guide as a basis for determining the control cards needed for executing and analyzing the program.

b. The job control cards required for loading and executing the program differ from the user's normal sequence as follows:

(1) The JAVS data collection routines must be loaded (JPROBESX). If the user's program is in overlay form, load JPROBESX in the main link.

(2) Load the instrumented object code instead of the original object code (in the appropriate link, if overlaid).

(3) Supply the AUDIT file (file code 08) for the execution trace results.

c. The JAVS post-test analysis (following user files and data) control cards include the AUDIT file (written during execution) and the JAVS data base library (LIBOLD on file code 01).

Figure G.1 shows the typical flow of operations in using JAVS. JAVS commands and the data base library are used in all activities except the compilation. The files used in the figure are the library (02,01), LPUNCH (07), object (C*), and AUDIT (08).

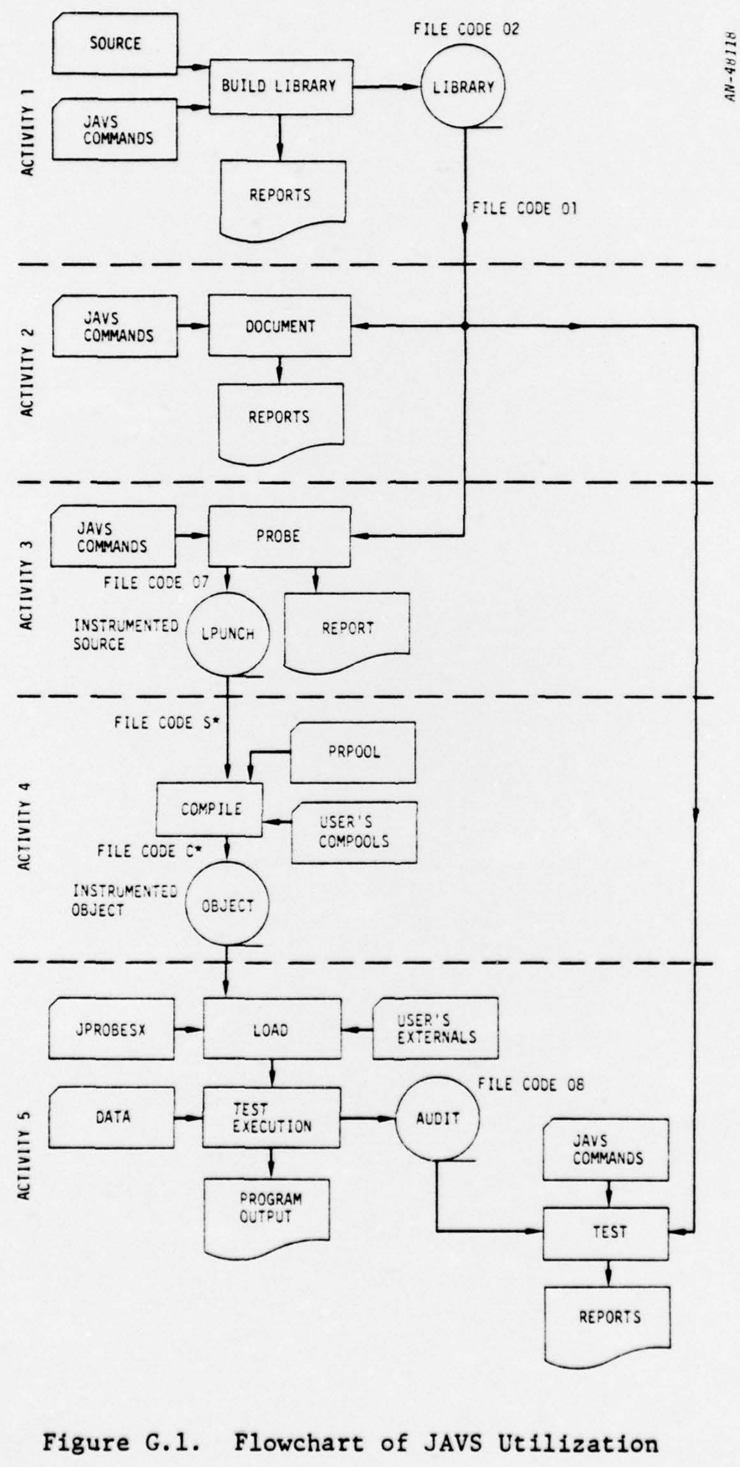


Figure G.1. Flowchart of JAVS Utilization

INDEX

INDEX (Cont.)

INDEX (Cont.)

REFERENCES

1. C. Gannon and N. B. Brooks, JAVS Technical Report, Vol. 2: Reference Manual, General Research Corporation CR-1-722, April 1977.

2. N. B. Brooks and C. Gannon, JAVS Technical Report, Vol. 3: Methodology Report, General Research Corporation CR-1-722, undated.

METRIC SYSTEM

BASE UNITS:

Quantity	Unit	SI Symbol	Formula
length	metre	m	...
mass	kilogram	kg	...
time	second	s	...
electric current	ampere	A	...
thermodynamic temperature	kelvin	K	...
amount of substance	mole	mol	...
luminous intensity	candela	cd	...

SUPPLEMENTARY UNITS:

plane angle	radian	rad	...
solid angle	steradian	sr	...

DERIVED UNITS:

Acceleration	metre per second squared	...	m/s
activity (of a radioactive source)	disintegration per second	...	(disintegration)/s
angular acceleration	radian per second squared	...	rad/s
angular velocity	radian per second	...	rad/s
area	square metre	...	m
density	kilogram per cubic metre	...	kg/m
electric capacitance	farad	F	A·s/V
electrical conductance	siemens	S	A/V
electric field strength	volt per metre	...	V/m
electric inductance	henry	H	V·s/A
electric potential difference	volt	V	W/A
electric resistance	ohm		V/A
electromotive force	volt	V	W/A
energy	joule	J	N·m
entropy	joule per kelvin	...	J/K
force	newton	N	kg·m/s
frequency	hertz	Hz	(cycle)/s
illuminance	lux	lx	lm/m
luminance	candela per square metre	...	cd/m
luminous flux	lumen	lm	cd·sr
magnetic field strength	ampere per metre	...	A/m
magnetic flux	weber	Wb	V·s
magnetic flux density	tesla	T	Wb/m
magnetomotive force	ampere	A	...
power	watt	W	J/s
pressure	pascal	Pa	N/m
quantity of electricity	coulomb	C	A·s
quantity of heat	joule	J	N·m
radiant intensity	watt per steradian	...	W/sr
specific heat	joule per kilogram-kelvin	...	J/kg·K
stress	pascal	Pa	N/m
thermal conductivity	watt per metre-kelvin	...	W/m·K
velocity	metre per second	...	m/s
viscosity, dynamic	pascal-second	...	Pa·s
viscosity, kinematic	square metre per second	...	m/s
voltage	volt	V	W/A
volume	cubic metre	...	m
wavenumber	reciprocal metre	...	(wave)/m
work	joule	J	N·m

SI PREFIXES:

Multiplication Factors	Prefix	SI Symbol
1 000 000 000 000 = 10^{12}	tera	T
1 000 000 000 = 10^{9}	giga	G
1 000 000 = 10^{6}	mega	M
1 000 = 10^{3}	kilo	k
100 = 10^{2}	hecto*	h
10 = 10^{1}	deka*	da
0.1 = 10^{-1}	deci*	d
0.01 = 10^{-2}	centi*	c
0.001 = 10^{-3}	milli	m
0.000 001 = 10^{-6}	micro	μ
0.000 000 001 = 10^{-9}	nano	n
0.000 000 000 001 = 10^{-12}	pico	p
0.000 000 000 000 001 = 10^{-15}	femto	f
0.000 000 000 000 000 001 = 10^{-18}	atto	a

* To be avoided where possible

☆U.S. GOVERNMENT PRINTING OFFICE: 1977-714-025/219